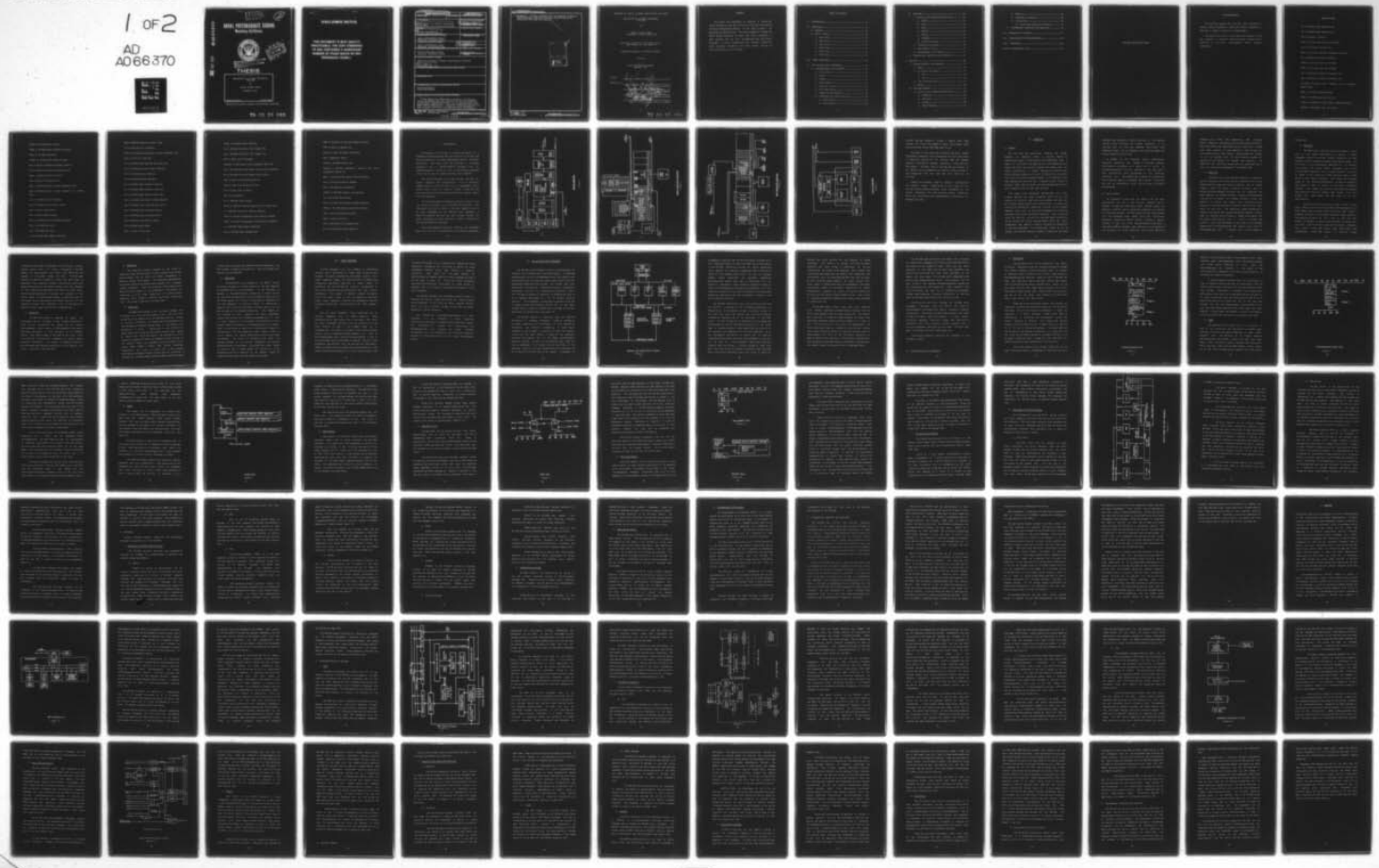
AD-A066 370 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 17/1
EVALUATION OF A SIGNAL PROCESSING TEST BED.(U)
DEC 78 G T VRABEL

UNCLASSIFIED NL

1 OF 2
AD
AD66370



NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

EVALUATION OF A SIGNAL PROCESSING TEST BED

by

George Thomas Vrabel

December 1978

Thesis Advisor: George Rahe

Approved for public release; distribution unlimited

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

(6) Evaluation of a Signal Processing Test Bed,

5. TYPE OF REPORT & PERIOD COVERED

(9) Master's Thesis; December 1978

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

(10) George Thomas Vrabel

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

(11) Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE

December 1978

13. NUMBER OF PAGES

107

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)

Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

(12) 108 p.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Array processing
Signal-processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis was undertaken to examine an acoustical signal processing test bed, similiar to the one installed at the Naval Postgraduate School, to be used primarily for experimental applications. The major components include two PDP-11 series computers, at least one array processor, a mass storage unit as well as assorted input and display

DD FORM 1 JAN 73 1473 (Page 1)

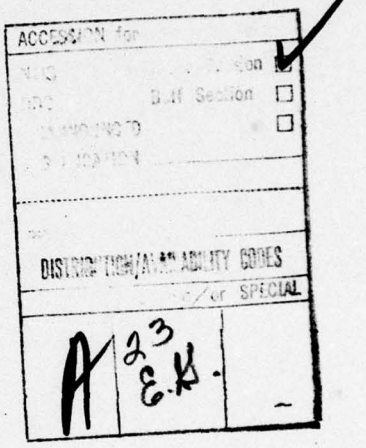
EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251 450

equipment. Of major interest were the computer selection, array processor selection and basic signal routing to facilitate real-time utilization



Approved for public release; distribution unlimited

EVALUATION OF A SIGNAL PROCESSING TEST BED

by

George Thomas Vrabel
Lieutenant, United States Navy

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1978

Author: _____

Approved by: _____
Thesis Advisor

_____
Second Reader

_____
Chairman, Department of Computer Science

_____
Dean of Information and Policy Sciences

ABSTRACT

This thesis was undertaken to examine an acoustical signal processing test bed, similiar to the one installed at the Naval Postgraduate School, to be used primarily for experimental applications. The major components include two PDP-11 series computers, at least one array processor, a mass storage unit as well as assorted input and display equipment. Of major interest were the computer selection, array processor selection and basic signal routing to facilitate real-time utilization.

TABLE OF CONTENTS

(This page intentionally blank)

ACKNOWLEDGEMENTS

The continual support and critical help provided by thesis advisor Professor G. Rahe and systems programmers A. Wong and W. Thomas is gratefully acknowledged.

The preliminary text of this thesis was prepared using the software of the UNIX operating system, operating on a PDP-11/50 of the Naval Postgraduate School Computer Laboratory.

ABBREVIATIONS

A1 ** AP-120B Adder Register One

A2 ** AP-120B Adder Register Two

A/D ** Analog to Digital

ADAM ** Analog Data Acquistion Module

ALU ** Arithmetic Logical Unit

ANSI ** American National Standards Institute

AP ** MAP-300 Arithmetic Processor

APGET ** Get Data From the AP-120B

APPUT ** Put Data Into the AP-120B

AP1 ** MAP-300 Arithmetic Processor One

AP2 ** MAP-300 Arithmetic Processor Two

AP-120B ** Floating Point Systems Array Processor Model 120-B

APAL ** AP-120B Cross-Assembler

APEX ** AP-120B Executive Routine

APDMA ** AP-120B AP Direct Memory Address Register

APLINK ** AP-120B Linker and Loader

APMATH ** AP-120B Math Library

APMAE ** AP-120B Memory Address Extension

APSIM ** AP-120B Simulator

APTEST ** AP-120B Path Tester Program

APS ** MP-300 Addresser Processor Section

APU ** MP-300 Arithmetic Processing Unit

CVMUL ** Complex Vector Multiply

CPU ** Central Processing Unit

CSPU ** MAP-300 Central System Processing Unit

CSW ** MAP-300 Control Status Register or C-State Word

CTL ** AP-120B Control Register

DCB ** MAP-300 Driver Control Block

DIO ** Direct Input/Output

DMA ** Direct Memory Access

DPA ** AP-120B Data Pad Address Register

DPX ** AP-120B Data Pad X

DPY ** AP-120B Data Pad Y

FA ** AP-120B Adder Result Register

FCB ** MAP-300 Function Control Block

FFT ** Fast Fourier Transform

FIFFT ** Forward/Inverse Fast Fourier Transform Test

FIFO ** First In First Out

FL ** AP-120B Adder Results Less Than Zero

FM ** AP-120B Multiplier Result Register

FMT ** AP-120B Format Register

FN ** AP-120B Function Register

FO ** AP-120B Adder Exponent Overflow

FU ** AP-120B Adder Exponent Underflow

FZ ** AP-120B Adder Results Equal Zero

HMA ** AP-120B Host Memory Access Register

HIC ** MAP-300 Host Interface Controller

HIM ** MAP-300 Host Interface Module

HIS ** MAP-300 Host Interface Scroll

IOS ** MAP-300 Input/Output Scroll

IQ ** MAP-300 Input Queue

LIFO ** Last In First Out

LITES ** AP-120B Lights Register

M1 ** AP-120B Multiplier Unit Number One

M2 ** AP-120B Multiplier Unit Number Two

MAP ** Macro Array Processor

MAP-300 ** CSPI Macro Array Processor Model 300

MD ** AP-120B Main Data Memory Output Buffer Register

MI ** AP-120B Main Data Memory Input Buffer

MOS ** Metalic Oxide Semiconductor

MTBF ** Mean Time Between Failure

MTTR ** Mean Time To Repair

NOP ** No Operation

OQ ** MAP-300 Output Queue

P0-P3 ** MAP-300 Program Counters One Through Three

P ** MAP-300 Multiplier Results Register

PIOC ** AP-120B Programmable Input/Output Channel

PIOP ** AP-120B Programmable Input/Output Processor

R ** MAP-300 Adder Results Register

RAF ** MAP-300 Read Address FIFO

RAMP ** Reliability And Maintenance Program

RFFT ** Real to Complex FFT

RFFTSC ** Real FFT Scale and Format

ROM ** Read-Only Memory

S-Pad ** AP-120B Scratch Pad

SNAP-II ** MAP-300 Systematic Notation For Array Processing Version II

SPFN ** AP-120B S-Pad Output Buffer Register

SRA ** Subroutine Return Address

SWR ** AP-120B Switch Register

SYSFLG ** MAP**300 System Flag Register

TM ** AP**120B Table Memory

TMA ** AP-120B Table Memory Address Register

TMRAM ** AP-120B Random Access Table Memory

VAC ** Volts Alternating Current

VMUL ** Vector Multiply

WAF ** MAP**300 Write Address FIFO

WC ** AP-120B Word Count Register

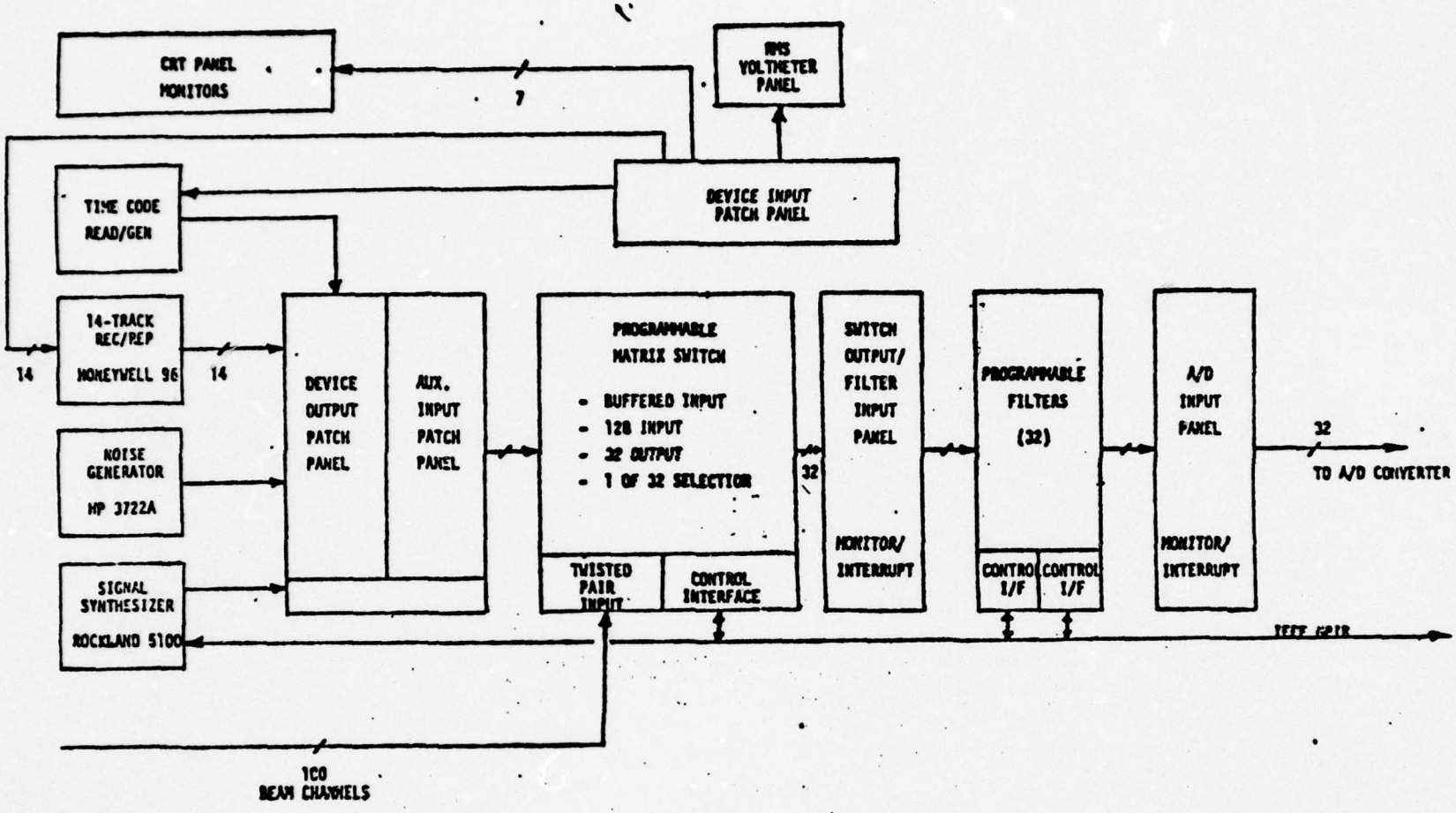
I. INTRODUCTION

The purpose of this study is to begin evaluation of a proposed signal-processing test bed similiar to the test bed being installed at the Naval Postgraduate School, Monterey, California. The basic test bed consists of an analog subsystem (fig 1), data-processing subsystem (fig 2), signal-processing subsystem (fig 3) and display subsystem (fig 4) to be used for general-purpose Naval research.

The analog subsystem of the test bed was designed for signal reception and conditioning. This is basically accomplished by a 128-line input into a programmed matrix switch which emits 32 lines of output. These 32 lines continue through a program-controlled filter issuing output from the subsystem.

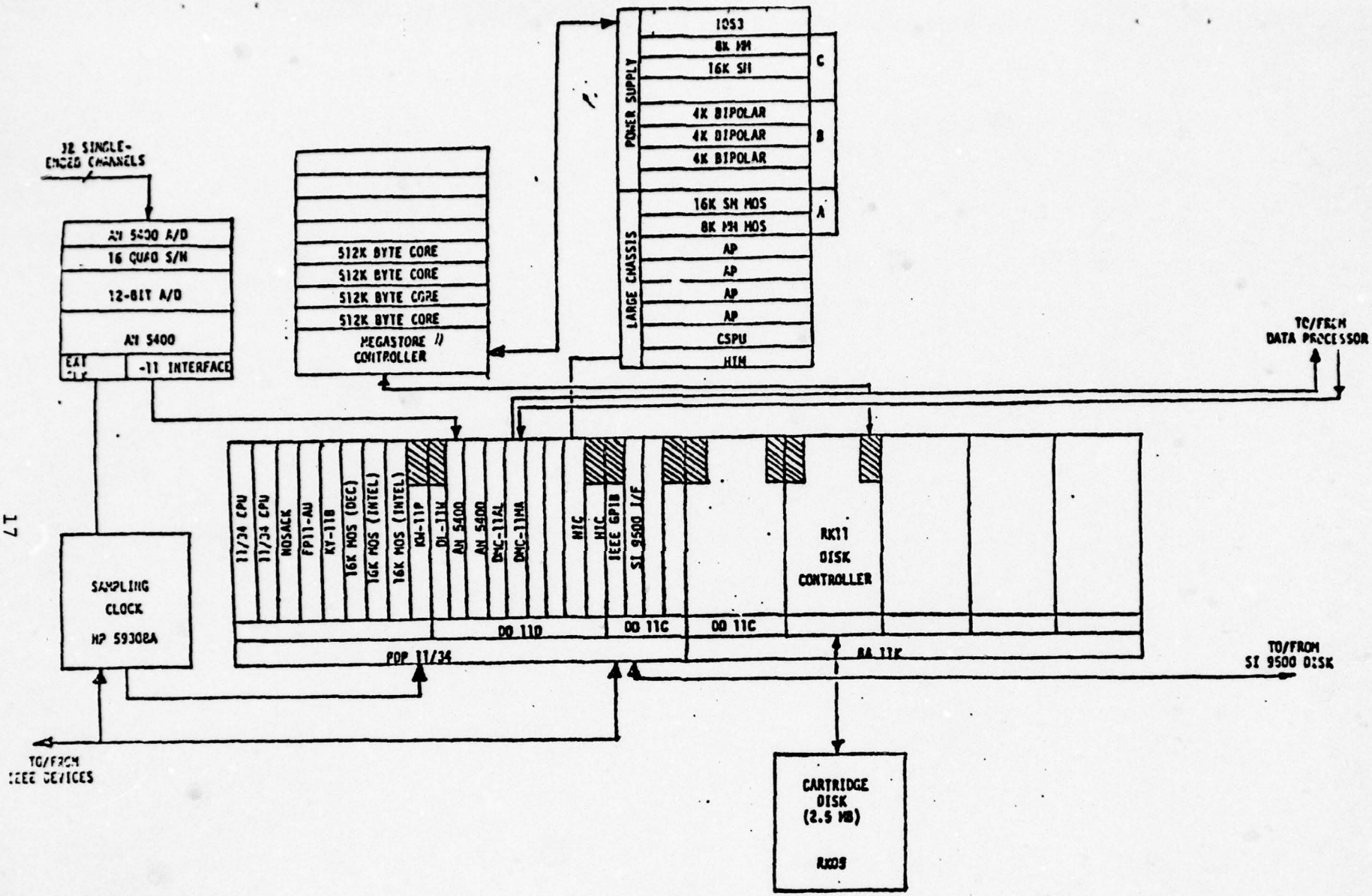
The signal-processing subsystem receives results from the analog subsystem via an AN-5400 A/D converter. This information can then be stored in an Ampex Megastore unit to be later processed by one MAP-300 array processor. A PDP-11/34 computer controls the mass storage device, the array processor and input functions. Output is directed to the data-processing subsystem.

The data-processing subsystem receives the processed data and controls the operation of the display subsystem.



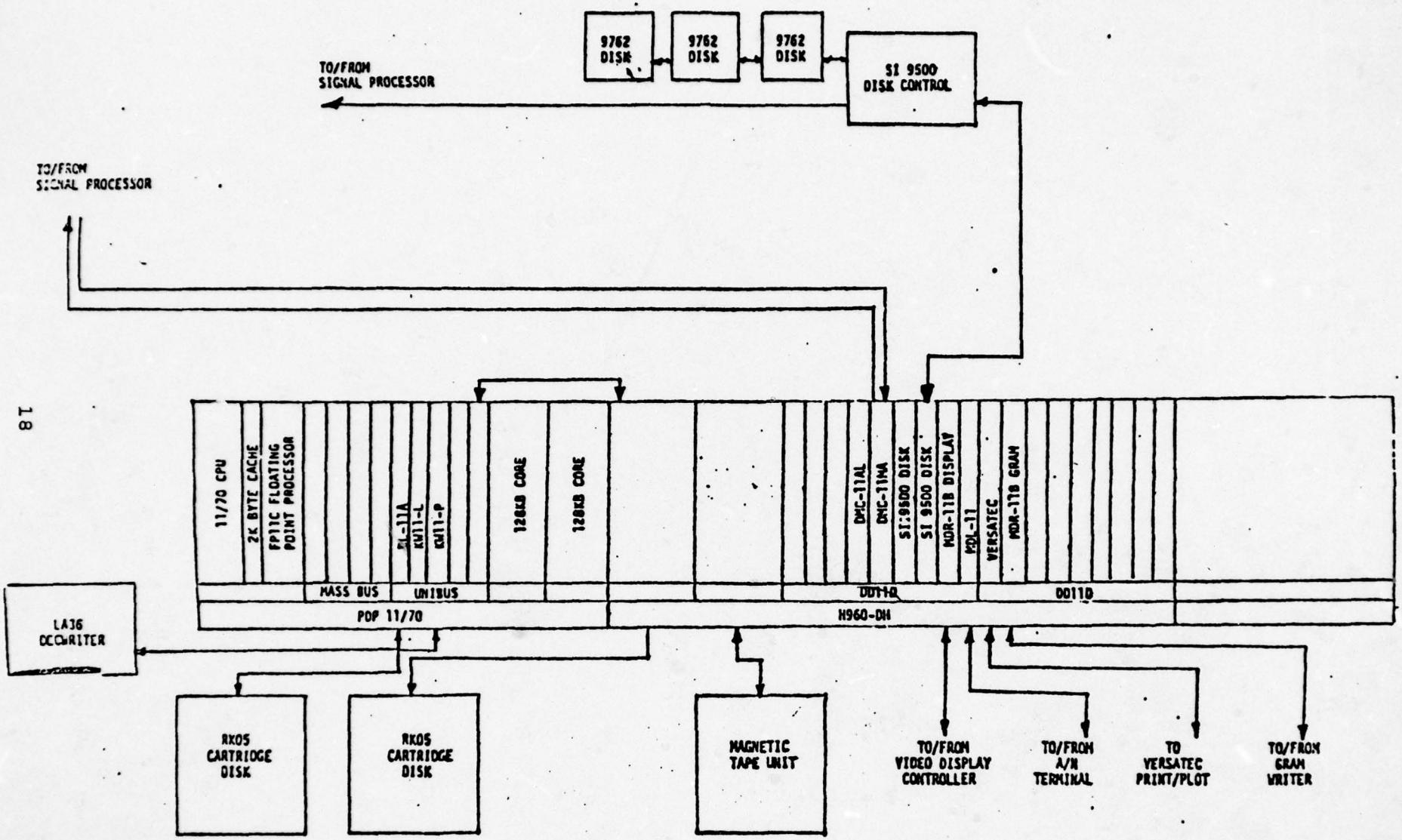
ANALOG SUBSYSTEM

Figure 1



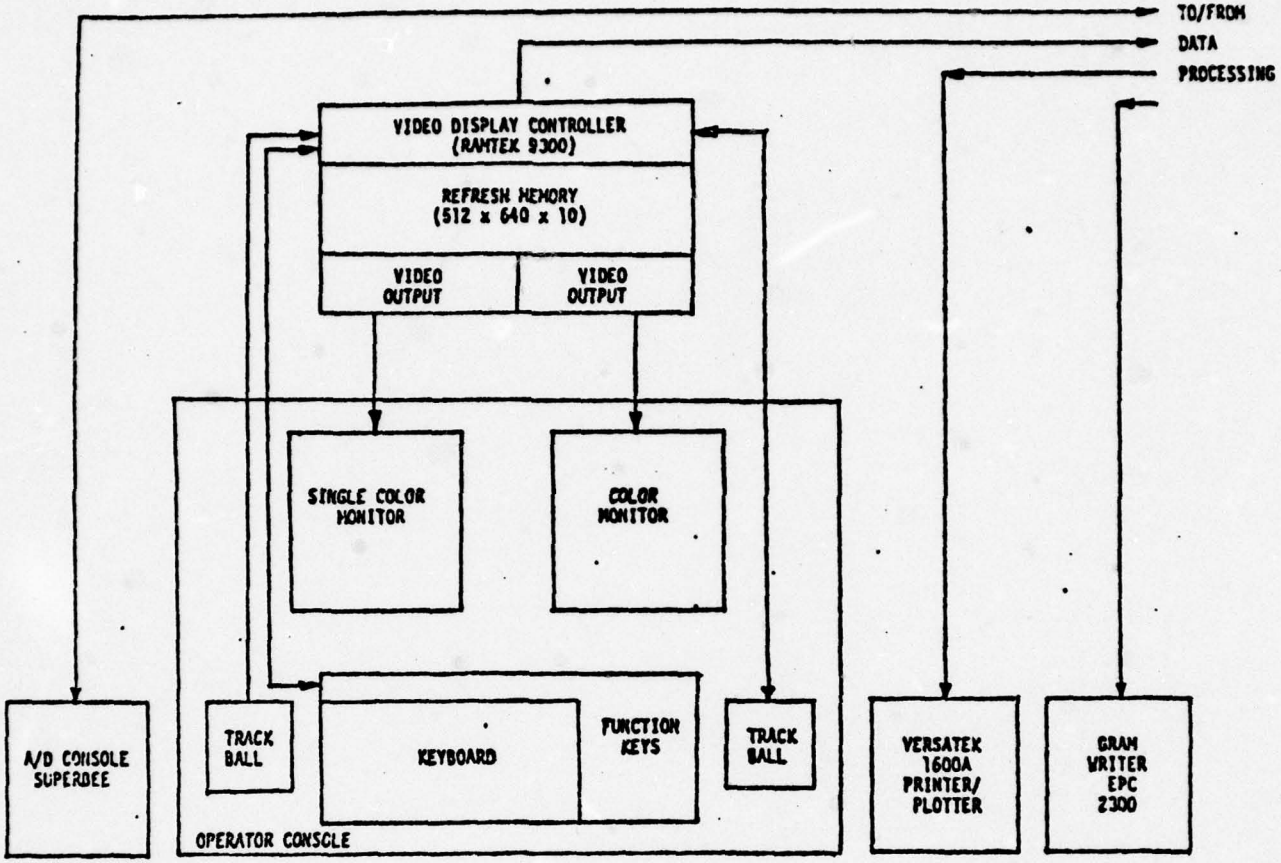
SIGNAL PROCESSING SUBSYSTEM

Figure 2



DATA PROCESSING SUBSYSTEM

Figure 3



DISPLAY SUBSYSTEM

Figure 4

Display devices presently include a Ramtek 9300 Video Display Unit (color and shades of gray), the Versatec 1600A printer/plotter and an EPC 2300 Gram Writer.

The goal of this study was to examine the major system components, computers, array processors and major data paths to determine feasibility for various uses and suggest possible alternative methods, especially in the real-time environment. The basic task of the test bed was assumed to be general with no suggestion of specific tasks although it was recognized that many uses and data rates may be utilized.

Chapter II discusses specific computer manufacturers and computer types. Chapters III, IV and V deal with the two most popular general-purpose array processors on the market, discussing the pros and cons of each. Chapter VI gives final conclusions and recommendations concerning the proposed test bed.

II. COMPUTERS

A. GENERAL

For the test bed evaluation, choosing the proper computer is important since a varying amount of computational power is required for each subsystem. Also, a gambit of functions and uses may be tried necessitating a system that must realistically emulate many speed, cost and memory constraints. A common and popular system affords better software support while still maintaining a low price. The ability to rely on system support is an important issue when considering long term use. A popular system tends to develop newer, more efficient software packages earlier and more frequently than do less used systems.

For large array processing applications with many display devices the ideal situation would be for one computer to initially load the array processor and then act as a "whole system" monitor and statistician. It could also perform the information gathering function while another computer would act as the output processor for the array processor and control the display devices. That situation would be similiar to that of a test bed where flexibility may be the key and being computer-bound would be highly undesirable and possibly unjustly influence the evaluation of the array processor. An ultimate goal might to be to choose the smallest computer capable of operating the array

processor and associated display devices in the desired fashion while providing for product expansion. It is realized that for test and research activities more computing power may be necessary than would be needed for normal production activities.

In October 75, the Computer Family Architecture Selection Committee was formed to evaluate computer architecture candidates as a basis for a family of software-compatible military computers. Ten Army and 17 Navy organizations were represented on the selection committee [11]. The purpose was to select an architecture which could be used as a standard, had a proven instruction set and an architecture which could be used in advanced technologies.

B. PDP-11 FAMILY

The committee voted that the PDP-11 had the best architecture for use in the Military Computer Family. However, it generally contained a small address space and possible floating point instruction compatability problems with existing systems. The IBM system 370 was ranked second with the Interdata 8/32 ranked third [12]. The Digital Equipment Corporation PDP-11 series provided a popular example of both the price and performance excellance in available computer systems. Their popularity is evidenced by the shipping of 10,000 PDP-11/04 and 10,000 PDP-11/34

computers as of 1975, 1976 respectively [28]. relevant PDP-11 computers considered were the PDP-11/04, PDP-11/34, PDP-11/45, PDP-11/55, PDP-11/60, and the PDP-11/70 (listed from least powerful to most powerful). What follows is a brief description of each system. Unless otherwise stated, it will be assumed that the more powerful system will contain all the features of systems less powerful. The PDP-11/03 and the LSI-11 series were not considered due to their not having the advantages of the UNIBUS [28].

1. PDP-11/04

The PDP-11/04 is the smallest computer of the PDP-11 series, containing the entire central processing unit on one board permitting room for drastic expansion due to unused chassis area. The system contains self-test logic to determine system operability every time the processor has power applied, the console emulator is used or the bootstrap routines are initiated. The console emulator allows the operator to control the system from a terminal without physically throwing switches or reading lights on the front panel of the unit. The bootstrap loader automatically restarts the system from various peripheral devices without need of physical switch throwing. Memory size varies from 8K bytes to 56K bytes (8 bits = 1 byte) of either MOS (metalic oxide semiconductor) or core type with an average access time of 500-nanoseconds and system cycle time of 725-nanoseconds [30]. A typical cost of this system is

$8,950 [29].

2. PDP-11/34

The PDP-11/34 is the next size of the PDP-11 family and is the lowest architecture to contain a memory management routine to provide program protection so user programs cannot access or change system memory space. (In the 11/04 it is the programmers responsibility to maintain and protect this area.) Memory management also allows virtual memory paging of up to 16 pages ranging in size from 64 bytes to 8K bytes for a total possible memory of 256K bytes of which 128K is physical. (The highest 4K of address space on the PDP-11/34/45/55/60/70 is used for registers that store I/O data or status of individual peripheral devices. This means that the 11/34 can physically address 124K bytes but virtually address 256K bytes.) The 11/34 allows both core memory and MOS memory to be used concurrently.

The PDP-11/34 also contains a memory option called cache memory which is a 2K high speed (300-nanosecond cycle time) memory used to store a copy of the the most recently selected portions of main memory affording faster access of instructions and data. The "hit" time or time the next access is resident in cache is approximately 86 percent for the 11/34. Time is saved by less area to access, therefore less search time, and shorter less complicated data transmission. Since MOS memory is volatile (loses

information when power is removed), the 11/34 has a battery back-up option which will retain information in the MOS memory for approximately two hours. The PDP-11/34 can operate in two modes, Kernel and User. This two mode concept is important in security since the User mode is prevented from executing certain instructions that could cause modification of the Kernel program, halt the computer or use memory space assigned to the Kernel or other users. Monitoring and Supervisory routines are executed in the Kernel mode. The Kernel/User concept is important since if the Kernel can be made secure, the overall security of the operating system from accidental harm is much easier to achieve. Prices range from $11,080 to $53,800 [29].

3. PDP-11/45

The PDP-11/45 system is designed for speed. The high-speed central processor allows program execution of three million instructions per second and has either 300-nanosecond bipolar memory or 980-nanosecond core memory available. MOS memory is also available as an "add-on" option. Total memory space is the same as the 11/34. There is an optional floating point processor to handle double precision arithmetic. The system is especially good for multiple-task applications, otherwise it is the same as the 11/34. The price is $41,800 [29].

4. PDP-11/55

The PDP-11/55 system improves on the 11/45 by inserting a dual bus structure to allow intermixing core and bipolar memory (up to 248K with memory management) to optimize system performance. Two separate semiconductor controllers allow simultaneous data transfer for increased system throughput. Both the 11/45 and 11/55 hardware have been optimized towards a multiprogramming environment by installing a third mode, Supervisor, to control system operation while properly handling multi-user operations [30]. The price is $50,400 to $80,780 [29].

5. PDP-11/60

The PDP-11/60 system is the interface between the mid-range mini and the more powerful mini. With the 11/60 we see the first capability to microprogram and four levels of priority interrupts. The system was also designed with the engineering trade-off between ease of maintenance and reliability in mind. A system that is very difficult to repair after failure may be less useful than an easy system to repair that fails more often. The availability of the system is a measure of mean time between failure divided by the quantity mean time between failure plus mean time to repair [MTBF /(MTBF + MTTR)] [30]. Digital Equipment Corporation has tried to allow for a more complex architecture (probable higher failure rate) by providing a Reliability and Maintenance Program (RAMP) software package

to help locate software and hardware errors, decreasing the MTTR thereby increasing availability. The price ranges from $42,400 to over $200,000.

6. PDP-11/70

The PDP-11/70 is the largest of the PDP-11 series and gives the power of a large computer at the cost ($63,000 to $144,880 [29]) of a minicomouter. It was designed to operate in high-performance systems and is ideally suited for real-time systems due to the high speed of execution and the 80-95 percent "hit" ratio of cache memory. Addressing of over four Megabytes of physical memory is theoretically possible with the 22 bit addresser, although 256K of this 4M must be used for the UNIBUS referencing. (The UNIBUS can only address 18 bits, therefore the memory management routine must convert the 4 Megabyte addresses as if it were a virtual location.) At the present time however only 2M of physical memory can actually be accommotated by the UNIBUS. There is the option to use 64 bit floating point numbers in calculations. With two megabytes of main memory there is little concern for memory constraints during a multi-task environment. The option of attaching high speed mass storage devices to the central processing unit through dedicated paths is available. The system has eight levels of priority and a large amount of flexibility in its programming making it possible to run several levels of display devices under varying loading conditions.

III. ARRAY PROCESSOR

An Array Processor is an unit capable of performing floating point operations on large data arrays or data streams. It usually operates as a peripheral device to a "host" computer system and best performs the repetitious reiterative operations requiring a large number of summations and multiplications typically encountered in matrix calculations such as correlations and fast fourier transforms. This system is special purpose and cannot "think" for itself since it has no executive functions except those necessary to control the mathematics required to perform additions, multiplications and data movement [18].

With an array processor, large transforms can be achieved dependent only on memory capacity. These transforms can be done faster than in the normal CPU since the array processor performs only one function at a time (here function is used in the broader sense as in transposition) and there is no need for the normal overhead control logic of a general purpose computer [8]. This is more advantageous than a special purpose computer in that an array processor can be programmed to execute various array processing applications and can also act as a peripheral. Ideally a system would be wanted that could handle any size arrays including the possibility of very large arrays if the

situation warranted. This is theoretically possible by using sequential processing and stringing a series of array processors together having each perform a specific operation. That would only be good, however, for applications not needing results of data processed in step N to be used in step N-1. Using one array processor, efficient and sufficient performance of large arrays is possible due to the special architecture and memory of the array processor.

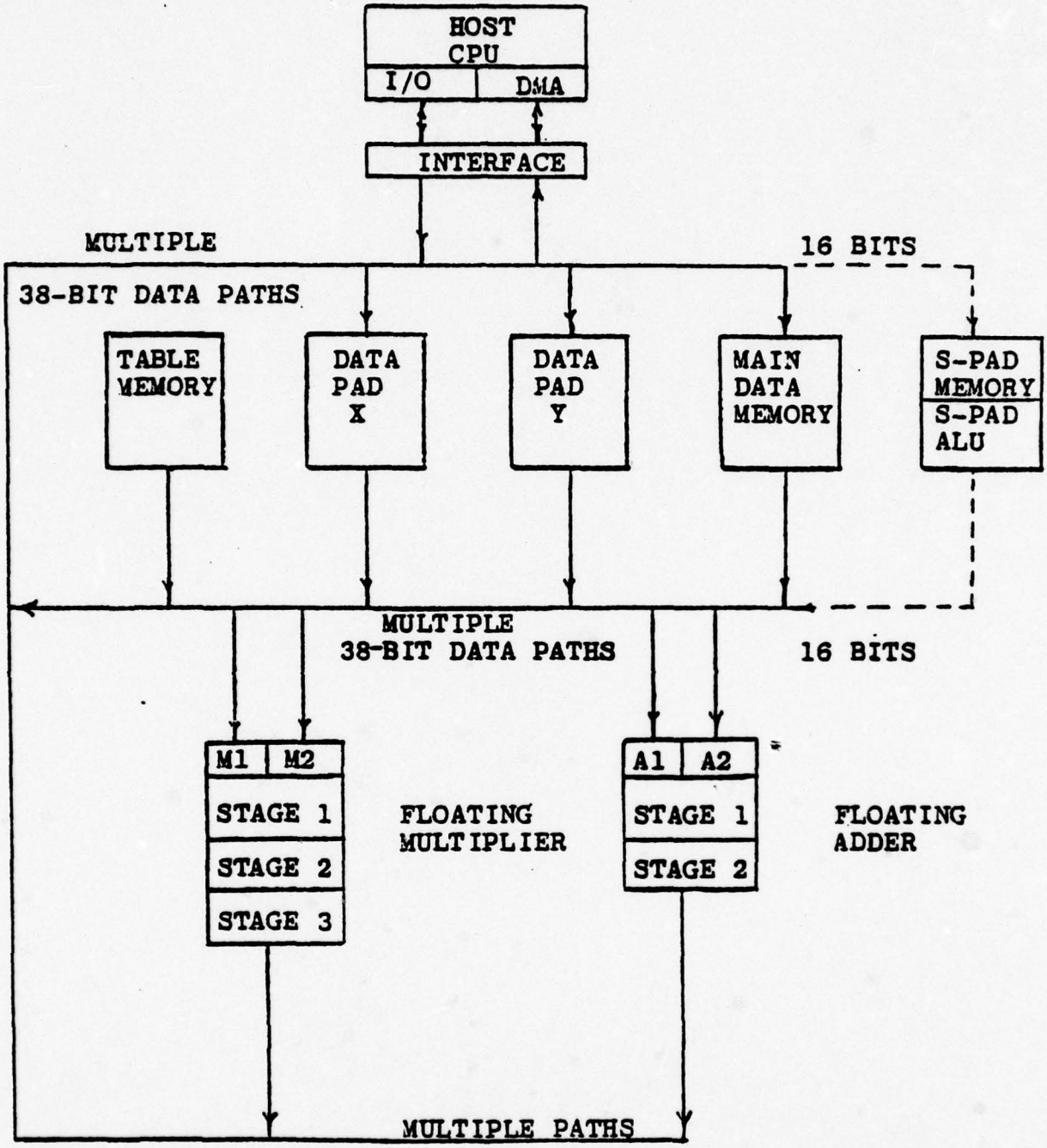
Two general purpose array processors presently seem to dominate the market. These are the CSP Inc. MAP-300 (Macro Array Processor) and the Floating Point Systems AP-120B. While the basic function of each is similiar, the actual operation is quite different.

The theoretical advantage/disadvantage of each processor will be discussed in detail comparing architecture, operational characteristics, software support and programability. Chapter VII, Conclusions and Recommendations, will discuss the actual problems encountered with the installation of the MAP-300 system to be used in the evaluation here at the Naval Postgraduate School.

IV. THE AP-120B ARRAY PROCESSOR

The AP-120B Array Processor (fig 5) is manufactured by Floating Point Systems Inc., Portland, Oregon. It operates synchronously using a 167-nanosecond cycle time master clock synchronized with a 50 percent safety margin every cycle for worst-case temperature and voltage. The system uses pre-conditioned medium-scale integrated circuitry, large-scale integrated circuitry and transistor-to-transistor logic. The AP-120B is capable of operating in temperatures from 10 to 40 degrees centigrade at 0 to 90 percent relative humidity. This processor is also able to operate using one of these various power options; 105/125 VAC at 120 amps, 180/228 VAC at 10 amps or 210/250 VAC at 10 amps with either 50/60 hertz or 50/400 hertz available [7].

The AP-120B employs a technique known as pipeline processing to increase throughput. Pipeline processing utilizes a combination of the elements of both sequential processing and parallel processing. A single basic processor, like an adder, is logically divided into integral units that can each perform a specific and separable function while another unit of the adder simultaneously performs another function of the addition task. When one task is completed, it will move on to the next step in the sequence allowing the just vacated section of the adder to be filled with the next task in the queue. Throughput is



General AP-120B Block Diagram

Figure 5

increased by insuring that the entire system is always full. This technique works with both the adder and the multiplier in the AP-120B. Pipelining is good for vector operations since vectors are basically independent and a solution of vector N is not needed before vector N+1 can be started. However scalar operations are basically sequential operations and cannot make use of pipelining [1]. By carefully considering every operation, especially those in loops, the programmer can squeeze more operations per time interval by pipelining than would be possible using standard sequential techniques. The time is generally limited by the multiplication time [14].

The AP-120B instruction word is up to 64-bits long and can perform a maximum of ten different operations in a single cycle. As an example, an add, a multiply, a move to and from each data pad (there are two) and an address increment or decrement can all be performed in the same cycle. Any one instruction or combination of the above can be performed as long as the resource required is not being used in another operation (some operations are multi-cycle and "lock-out" the resource until they are complete). It is the programmers obligation to insure that all required resources are available when they are requested or else they will be lost [7]. As an example; a read from a data pad takes at least two cycles. If cycle N wanted to read from Data Pad X and cycle N-1 already initiated a read from Data Pad X, the entire instruction word for cycle N would be

delayed one cycle waiting for the resource to become available. This ability to perform more that one basic operation per cycle allows a theoretical 30 million instructions per second to be executed. Due to memory size limitations and algorithms not needing ten operations per instruction word for sustained periods this rate can never be fully attained except possibly for short bursts [36]. Since some of these operations are housekeeping functions, the maximum number of arithmetic operations per second theoretically possible is twelve million for vectors and five million for scalars (scalar speed is much lower since it requires sequential processing and cannot take advantage of pipelining) [1].

The AP-120B uses a 38-bit data word which Floating Point Systems contends generates better accuracy than the 32-bit word commonly used by other systems [7]. This 38-bit word consists of a ten-bit biased exponent and 28-bit twos compliment mantissa thereby allowing numbers in a range of 3.7 * 10 ** -155 to 6.7 * 10 ** 153 to be represented. The 28-bit mantissa allows for extensive calculations without significant truncation errors or a maximum relative error of approximately 7.5 * 10 ** -9 per arithmetic operation or about 8 decimal digit accuracy. Floating Point Systems Inc. also employes a technique known as convergent rounding which they assert forces the roundoff error to approach zero.

The AP-120B does not contain the normal bus structure of other array processors but instead uses dedicated 38-bit data paths for the movement of data. There are two paths available to the adder (one for each input register), two paths to the multiplier and three paths available to the memory and data pads. This allows seven independent data words to be transferred each cycle. (This coupled with an add, multiply and address increment/decrement, equals the ten instructions per cycle possible.) These separate data paths eliminate the need for a handshaking arrangement between logic elements, although hankshaking is required when the AP-120B communicates with the host [7,36].

The price of a unit which includes the AP-120B array processor, interface with the PDP-11, 16K words of 333-nanosecond interleaved MOS memory, expansion chassis, installation, 256 words of program source memory, 512 words of Read Only Memory (ROM) table memory, a linker, loader, simulator, debugger, algorithm library and executive is $50,970.00 [10]. This includes a 90-day warranty with a servicing agreement available at extra cost. The field test mean time between failure is 3500 hours [3].

The following section explains the hardware of the AP-120B in detail.

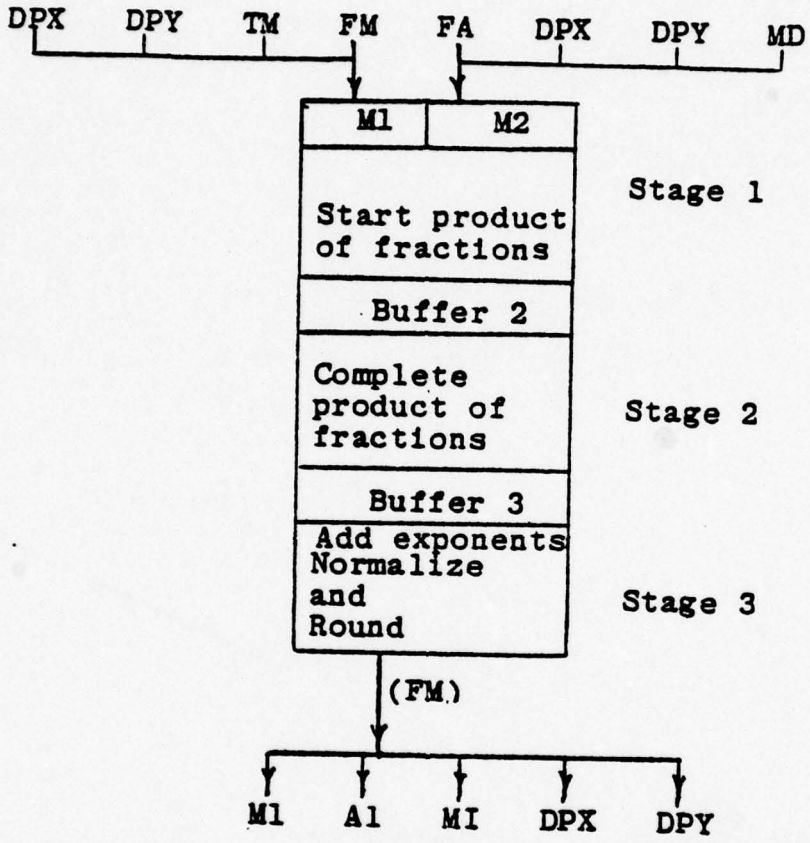
A. CHARACTERISTICS AND HARDWARE

1. Multiplier

The Multiplier unit (fig 6) consists of two 38-bit multiplier registers M1 and M2, three multiplication stages and a 38-bit register to store the result (FM). To receive a resultant after initiating the multiply, three cycles or 500-nanoseconds are required. Inputs to the M1 register can come from Data Pad X (DPX), Data Pad Y (DPY), Table Memory (TM) or the Multiplier result register (FM). Inputs to M2 are either from DPX, DPY, Adder result register (FA) or Main Data Memory Output Buffer (MD). Results from the multiplier can go to M1, the Adder input register (A1), Main Data Memory input buffer (MI), DPX or DPY.

Stage one of the multiplier starts the product of fractions by beginning the multiplication of the two 28-bit mantissas. This multiplication is completed in stage two resulting in a 56-bit mantissa. Stage three adds the exponents as it normalizes and convergently rounds the 56-bit mantissa to 28-bits. This stage also detects exponent overflow/underflow and if either exist will set the FO of FU bit in the status register. The status register can be read by the program to determine; if conditions are met from an arithmetic operation, to specify errors, or to be used in branching logic. These bits are available for testing one cycle after completion of the multiply.

This three stage multiply allows pipelining to be used since each stage is independent of the other two which



Floating Multiplier

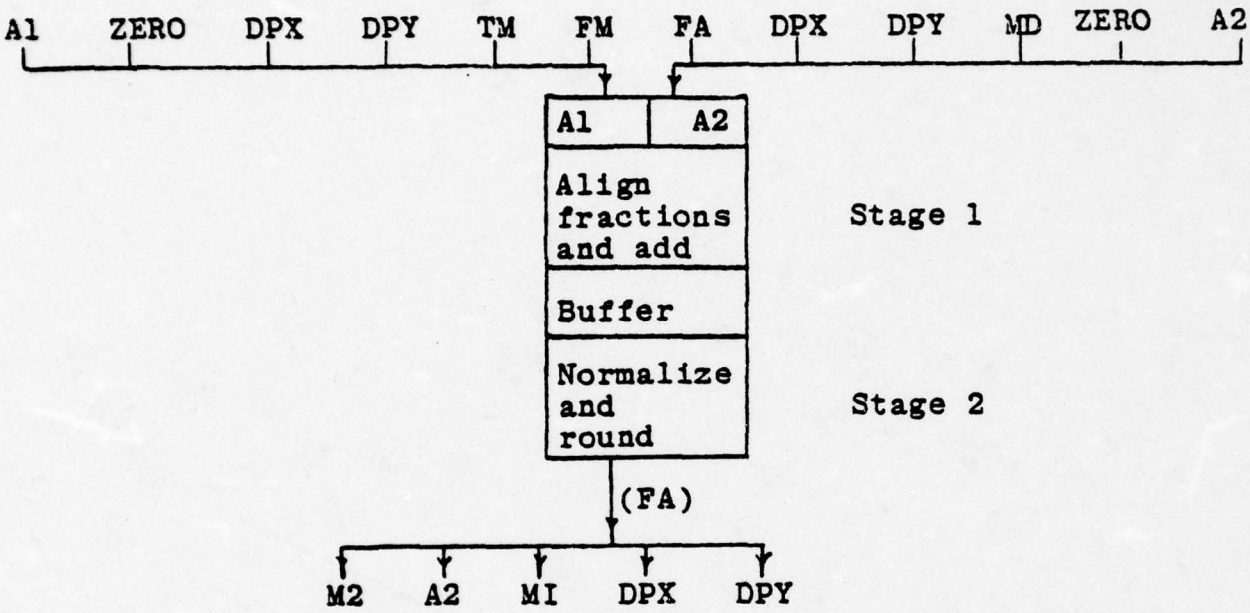
Figure 6

permits a multiplication result to be present at the result register every 167-nanoseconds once the pipeline becomes full (three cycles required to fill). Note that 500-nanoseconds are required if the result of the multiplication is required in the next multiplication as is the case with scalar arithmetic.

A readily apparent problem with the multiplier is that M1 receives inputs from both the Table Memory (TM) and the Multiplier Result register (FM) while M2 receives inputs from neither. Therefore, if a constant from TM were to be multiplied by the result of a just-completed multiplication, it would require an extra two cycles since either FM or TM would first have to be written into DPX or DPY and then written into M2. This disadvantage is overshadowed by the fact that even though dedicated data lines cause the above problem, in most cases they present a distinct advantage by allowing multiple data transfers in any given cycle [32].

2. Adder

The operation of the adder (fig 7) is similiar to that of the multiplier and consists of two 38-bit adder registers A1 and A2, two adder stages and an adder result register (FA). The addition of two numbers requires 333-nanoseconds (two cycles). Inputs to A1 are from Table Memory (TM), Multiplier Output register (FM), Data Pad X (DPX), Data Pad Y (DPY) and the ZERO constant while inputs to A2 are from the Adder Output register (FA), Data Pad X



Floating-Point Adder Unit

Figure 7

(DPX), Data Pad Y (DPY) and the ZERO constant. The results from the adder can go to A2, M2, DPX, DPY or MI. Stage one aligns the mantissas by shifting the smaller value, based on the value of the exponent, to the right until both exponents are equal then adding or subtracting these mantissas. Stage two normalizes and convergently rounds the mantissa and adjusts the exponent. This stage also sets four bits in the status register to denote results equal zero (FZ), results less than zero (FL), exponent overflow (FO) or exponent underflow (FU). These bits may be tested by other program instructions one cycle after the addition is completed. (Note that FO and FU are the same bits that are set by the multiplier on exponent overflow or underflow.)

As with the multiplier, the two-stage adder allows pipelining and a result can be generated every 167-nanoseconds. The adder does not have the disadvantage of inputting Table Memory (TM) values at the same register as FA but does have the multiplier result FM at the same adder input register (A2) as TM values. There is therefore not the ability to immediately add a FM value with a TM value without first going through DPX or DPY [32].

For both the adder and the multiplier there would be a two cycle time loss if FM was just loaded with a new value from the multiplier when it was needed for the addition/multiplication process (time N) and only a one cycle loss if it was ready the cycle before needed (time N -

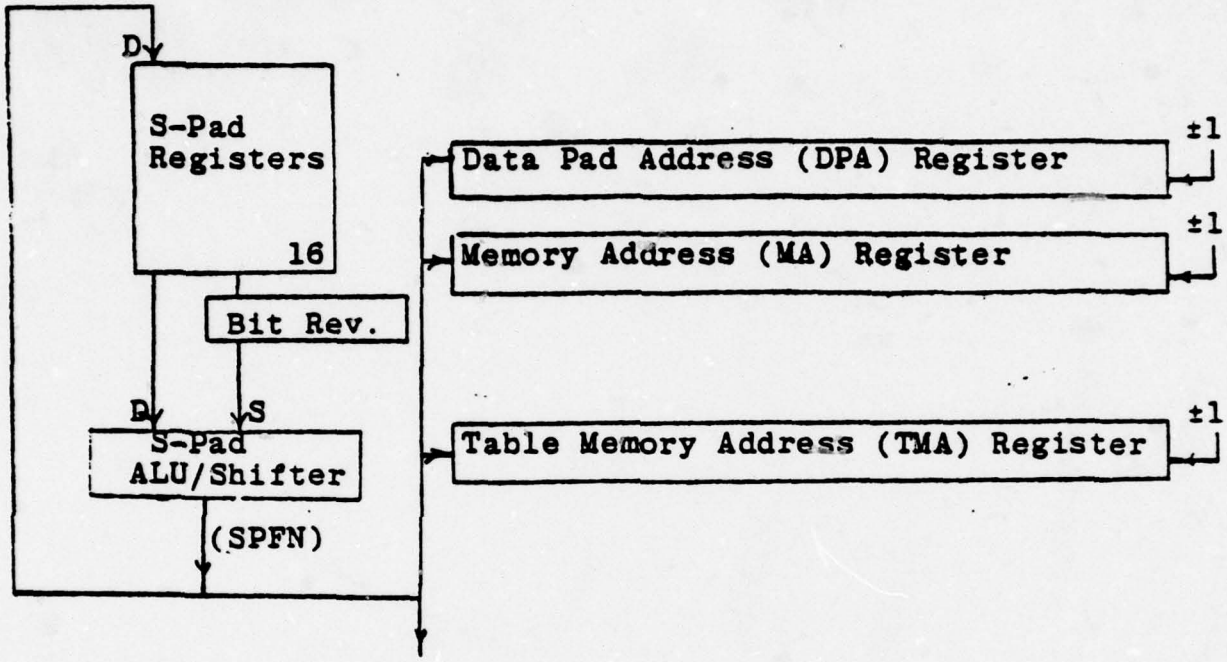
1 cycle). Otherwise there would be no loss of time since steps could be taken to move the value in FM through the DPX or DPY which would make it be available at the adder/multiplier input register when necessary. (Presupposing of course that the data paths to or from memory were not needed for other uses.)

3. S-Pad

The S-Pad (fig 8) (pseudonym for scratch pad) consists of the S-Pad Memory, S-Pad Arithmetic Logical Unit (ALU), Data Pad Address Register (DPA), Memory Address Register (MA) and the Table Memory Address Register(TMA). The sole purpose of the S-Pad is to compute addresses for Table Memory, Main Data Memory and the Data Pads. The S-Pad can operate concurrently with the memories, Multiplier and Adder [7].

The S-Pad Memory is made up of 16 registers each 16 bits wide giving the ability to compute an effective address of 64K. These registers may be assigned label names like "pointer" by the use of pseudo-operators, to make programs more readable, or may be directly addressed by number.

The S-Pad Arithmetic Logical Unit forms the operand addresses and also automatically loop counts, shifts the addresses left once (divide by two), shifts the addresses right once (multiply by two) or right twice (multiply by four). There is also the ability, if required, of bit



S-Pad Unit

Figure 8

reversal, to swap bits while accessing data in a scrambled order after a Fast Fourier Transform. The results of the S-Pad arithmetic logic unit, called SPFN, set bits in the status register to indicate whether the results were less than zero (N), zero (Z) or if there was a carry bit (C). These bits are available for testing by program instructions at the next instruction cycle.

TMA, DPA and MA store the computed address from the S-Pad ALU. The contents of each can either be changed by the value of SPFN or incremented by one. One cycle is required to compute the address and load it into the proper register [32].

4. Table Memory

Table memory is a 512 word, 38-bits per word bipolar read-only memory used to store important and much used constants. This memory has a 167-nanosecond cycle time but requires two cycles to get the value from memory to the output register TM [7]. Values in TM are available for use by DPX, DPY, MD, M1 and A1. These values may be requested every machine cycle and are initiated by changing the contents of the Table Memory Address Register (TMA) in the S-Pad. The programmer must control the timing necessary to insure the correct constant is at TM when needed due to the 2 cycle access time requirement.

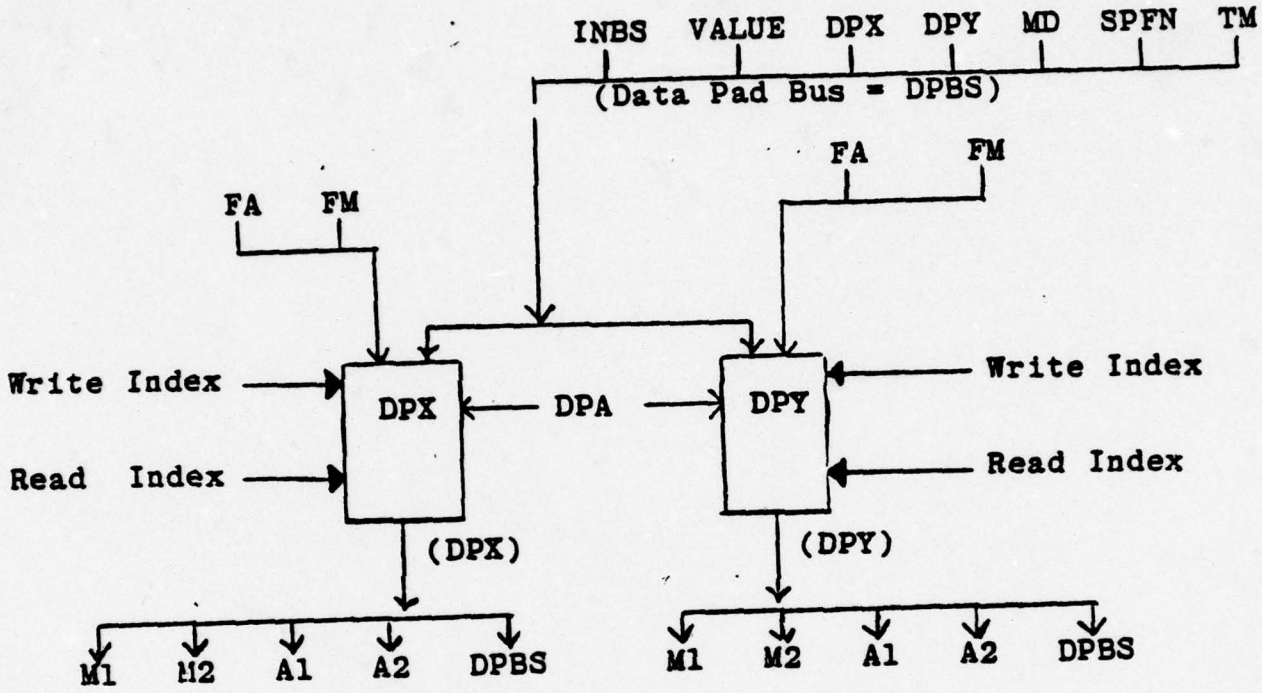
In the Fast Fourier Transform Mode, the address in TMA is interpretted by the hardware to be the angle which points to the appropriate root of unity for a particular step in the FFT alogrithm. Therefore, in a single quadrant of cosines, a full table can be represented [32].

There is an optional Random Access Table Memory (TMRAM) containing 1K of random access memory [8]. This allows loading of special constants necessary for special applications without the overhead of computing them every time or using valuable data pad space to store them. The price of this option is approximately $1850.00 [7].

5. Data Pad X and Y

The Data Pads (fig 9) consist of sixty four 38-bit accumulators, four of which are available from the 16 addressable each instruction cycle [7]. These 64 accumulators are divided into two 32-register blocks called Data Pad X (DPX) and Data Pad Y (DPY). From each Data Pad, one register can be read and another written during the same cycle.

The restrictions are that the same register cannot be read and written simultaneously and that a read and write operation during the same cycle must occur on registers whose addresses differ by no more than 7 due to base-address-plus-offset addressing. (However a register in DPX may be written at the same time as a register in DPY even if



Data Pad

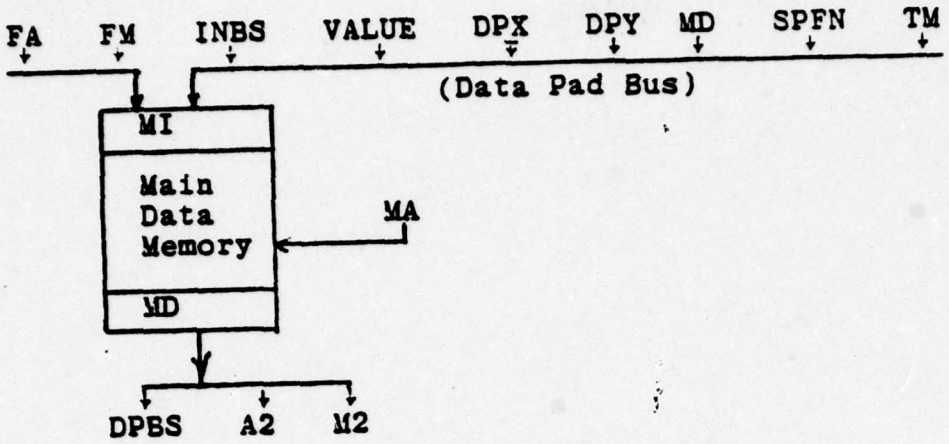
Figure 9

they both have the same address.) In the S-Pad, the Data Pad Address Register (DPA) supplies the base address to be used by the read/write instruction to locate the proper Data Pad register. The DPA supplies both DPX and DPY concurrently. The instruction uses this base address and an offset in the form DPX(offset) or DPY(offset) and can address -4 to +3 offset from the base in each Data Pad to find the effective address. Therefore if the DPA contains decimal value 20, registers 16, 17, 18, 19, 20, 21, 22 and 23 can be addressed in each data pad. The register addresses of both Data Pads range from 0 to 37 (base 8) and are arranged in a circular addressing scheme. Therefore 37 (base 8) +1 = 0 and the programmer need not be concerned about writing into a non-existant location but must only be concerned with overwriting previously written information.

DPX and DPY receive information from MD, FA, FM, DPX, DPY, output of the S-Pad arithmetic logical unit (SPFN) and VALUE (an immediate value used by immediate instructions arriving from the command buffer). DPX and DPY supply values to M1, M2, A1, A2, DPX, DPY and MI [32].

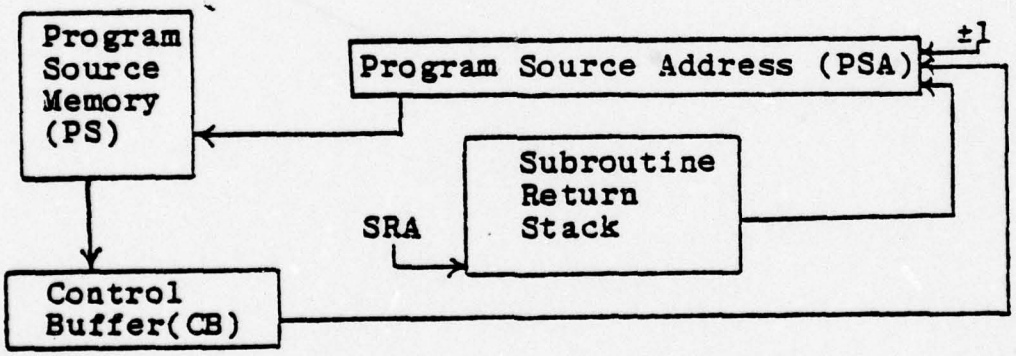
6. Main Data Memory

Main Data Memory (fig 10) contains 64K 38-bit words used primarily to store inputted data which will be operated on by the program. This memory is available in two forms, 167-nanosecond hardware interleaved MOS with 4K word segments or 333-nanosecond hardware interleaved MOS with 8K



Data Memory Unit

Figure 10



Control Unit

Figure 11

word segments. Both memories have a two bit parity option available [7] and a one megaword page selection option [9]. With memory limited to 64K, the largest complex-to-complex Fast Fourier Transform possible is 32K, which may not be acceptable in some applications.

Main Data Memory receives input information into its Memory Input Buffer (MI) from FA, FM, MD, DPX, DPY, TM, SPFN and VALUE. It can output via the Memory Data Buffer to DPX, DPY, A2 and M2.

Memory read or write may be requested every other cycle by changing the value of the Memory Address Register (MA) in the S-Pad. This yields an effective memory cycle time of either 333-nanoseconds (167-nanoseconds plus one machine cycle) or 500-nanoseconds (333 plus one machine cycle) dependent on the type of memory installed [32]. By special programming techniques and proper chip procurement, this overhead can be reduced to the advertised memory speed with the restrictions that the memory alternate between chips or alternate between even and odd boundaries. If effective speed is essential, it becomes the programmers responsibility to insure data location is known to the program at all times[8]. A read requires three cycles for information to be present in the MD if using 333-nanosecond memory and two cycles if using 167-nanosecond memory. This information will be available until a new value overwrites it. If a write or read is initiated before two memory

cycles (unless special chips and techniques of above are used), the request will not be lost but the memory will automatically provide a hardware lockout (wait until memory available for read/write) [14].

The value in the Memory Address Register (MA) points to the desired location in main data memory. MA may be either set to a specific value or incremented/decremented by one in the S-Pad. Since there is a slight time lag between when a value is requested to be placed in MD and when it actually gets there, the programmer must always be aware of what values are in MI and MD, to allow the proper "set up" time to get these values to either the Adder, Multiplier or correct DPX, DPY or MI address [32].

7. Program Source Module

The Program Source Module (fig 11) consists of the Program Source Memory (PS), Program Source Address Register (PSA), Control Buffer (CB) and the Subroutine Return Stack (SRS) [32].

The PS is a high speed, 50-nanosecond, bipolar memory addressable to 7K 64-bit words and is available in 256 word increments [4]. The PSA contains the address of the next instruction and is incremented by one after instruction execution unless modified by either the Control Buffer (new address as a result of a branch or jump instruction) or the Subroutine Return Stack. The SRS saves

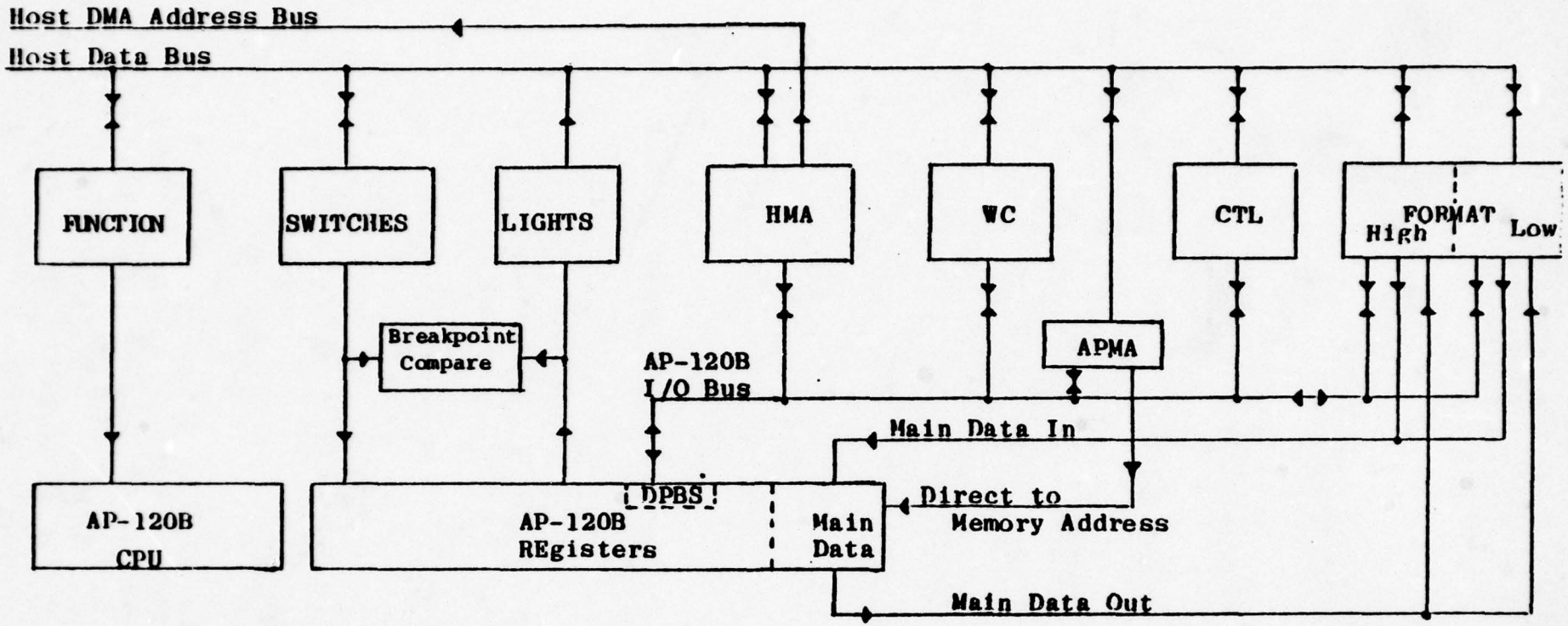
the current PSA when a Jump Subroutine instruction is performed and increments the value of the Subroutine Return Address (SRA). When a Return instruction is performed, the SRA is decremented by one making nested subroutines possible. The Control Buffer decodes and executes the instruction as the CPU would in a general purpose computer [32].

8. Interface with PDP-11 Series

The interface unit with the PDP-11 series contains two major segments, the Front Panel and the DMA Controller and Formatter. The Front Panel contains three registers and is used mainly as a debugging aid while the DMA Controller and Formatter contains five registers and is used for program and data entry or removal.

a. Front Panel

The Front Panel (fig 12) consists of three 16-bit registers, the Switch Register (SWR), the Lights Register (LITES) and the Function Register (FN). The Front Panel is used for bootstrapping and debugging of user programs. These three registers can be examined by the host and take the place of the toggle switches normally on the front panel of the console [32]. With the use of the Debugger program, these registers can effectively breakpoint the AP-120B at a selected program location or data address. This Front Panel allows each program to be single-stepped



AP120-B PANEL AND HOST INTERFACE

Figure 12

through its execution sequence [6,7].

The Switch Register is written by the host computer but can be read by both the AP-120B or the host. The SWR is used to enter data and addresses into the AP-120B, primarily for debugging. Its contents can be fed to the DPX, DPY, MD or the S-Pad.

The Lights Register simulates the front panel lights of the console. This register is set by the AP-120B and can only be read by the host. LITES is used to display selected contents of the internal registers of the AP-120B.

The final register is the Function Register which provides front panel toggle-like controls to the AP-120B. The FN can stop, start, step or reset the AP-120B. It can also continue operation resuming at the current value of the PSA, examine a register, examine a portion of a register or memory contents of a selected area, deposit the contents of SWR into a selected register or memory location and then breakpoint according to the values of TMA, MA or DPA. The FN can also increment the TMA, MA or DPA after completion of an instruction to facilitate stepping through memory locations [32].

The Front Panel is advertised to be invaluable in troubleshooting when used in conjunction with the interactive Debugger routine.

b. DMA Control

The DMA Control is the second half of the interface and consists of three 16-bit registers, one 18-bit register and one 38-bit register. DMA Control is responsible for transferring programs and data between the AP-120B and the host computer. This section of the Front Panel will also do format conversion "on the fly" which should effectively alleviate time lags [32]. Four types of data transfer combinations are possible, host DMA to AP-120B DMA, host DMA to AP-120B Programmed I/O, host Programmed I/O to AP-120B Programmed I/O and host Programmed I/O to AP-120B DMA with a maximum theoretical burst transfer rate of three megawords per second for all types of transfers [7].

The Format Register (FMT) is a 38-bit double-buffered register used to perform all transfers of floating-point numbers from the host to the AP-120B [32]. The FMT will convert 16-bit integer numbers to 38-bit unnormalized floating-point numbers, 32-bit PDP-11 integers to 32-bit AP-120B integers and 32-bit floating-point numbers to 38-bit floating-point numbers. All these operations are in reverse for the AP-120B to host direction [7]. Since the PDP-11 is a 16-bit computer, it will access the Formatter in 16-bit half-words to be compatible. It must be noted that for some applications, such as difference filtering, there is a possiblity of extreme accuracy loss due to 16-bit integer to 38-bit floating-point conversion. The synthetic

precision generated by such a conversion can cause certain coeffiecient combinations, such as +1 and -1, when multiplied by mirrored arrays, to result in errors when reconverted to 16-bit format. The programmer must be aware of these possible losses and test for them before faith is placed in the result.

The AP Direct Memory Address Register (APDMA) points to consecutive locations in AP-120B Main Data Memory during DMA transfers. This register can be automatically incremented/decremented allowing blocks of information to be read into consecutive locations with minimal overhead.

The Host Memory Access Register (HMA) operates similiar to the APDMA except it points to consecutive memory locations in the host memory. In the PDP-11 this memory is 256K so the HMA is 18-bits to allow for this addressing capability.

The Word Count Register (WC) counts the number of words transferred during a DMA operation. This register must be preset to the required number of words and will stop DMA transfer when the prescribed number of words is transferred.

The final and most important register in the interface is the Control Register (CTL). It controls the direction and mode of transfer, type of format conversion and provides certain status bits pertaining to the transfer.

This register, with the use of HMA and/or APDMA, allows the host to execute other programs and be interrupted when the DMA is completed. This CTL also allows either the host or AP-120B to control the data transfer. (The AP-120B must control transfer from a loaded program since the executive alone is not powerful enough to control data transfer [32].)

B. SOFTWARE

Various software support, executive and development programs are available with the AP-120B.

1. Executive and Associated Routines

The AP-120B provides executive and housekeeping routines to increase the effectiveness of operation and enhance program development.

a. APMATH

APMATH is a series of approximately 150 [8] library functions, vector and matrix subroutines and signal processing algorithms [7] written in AP-120B assembly language [8]. These routines are callable from either host Fortran, host Assembly or AP assembly languages [36] with the use of the AP Executive. These programs can reduce the run time and decrease programming time by presenting some of the most common array processing functions in subroutine callable form. These routines include: data transfer and control; basic vector arithmetic; matrix operations and Fast

Fourier Transform; all of which are able to work with both real and complex data.

b. APEX

APEX is the AP Executive routine which is resident in the host computer and allows the AP-120B to communicate with the host computer via Fortran or host Assembly language calls. APEX decodes subroutine calls from the host computer [36] and directs the AP-120B to perform the specified action. Both APMATH routines and user written routines may be called by the AP-120B from the host computer [32].

c. APAL

The AP Cross Assembler (APAL) is a two pass assembler written in Fortran IV which requires 24K memory in the host computer to operate. APAL assembles source text written in AP Assembly language into object code understandable by the AP-120B. The assembler also optionally produces an AP Assembly listing containing errors in both passes, location counters, assembled data, the symbol table and source statements.

APAL recognizes signed constants ranging from -32768 to 32767 and unsigned constants from 0 to 65535 both of which may be represented in binary, octal (default base), decimal or hexadecimal. It allows free formatting but recognizes the general source statement form: optional

label followed by a colon, multiple op codes separated by semicolons (one to ten operations which total no more than 64-bits. Sixty four-bits is the maximum dictated by seven data transfers, one add, one multiply and one address increment/decrement), and an optional comment statement denoted with leading double quote (").

Once the modules are written, APAL can be operated dynamically, allowing the programmer to build the program at assembly time. APAL will question the operator about the source file name, destination file name etc. and subsequently will prompt him concerning missing items. If there are errors in the module, these can be changed dynamically without reassembling the entire module [4].

d. APLINK

The AP linker (APLINK) is written in Fortran IV and requires approximately 10K of memory in the host computer. APLINK performs functions similiar to those of any other link editor which include relocation and assigning absolute addresses to the object module, correlation of global entry symbols in one module with external symbols in the other modules, loading the module from the program library and production of the final load module. These functions are performed interactively with dialogue between APLINK and the user at the console.

Besides linking the modules, APLINK returns to the console any symbols in a file which are undefined, will output the symbol table and locations when requested and returns the high address and starting address to be used with the Debugger routine [5].

e. APSIM

APSIM is the AP-120B simulator and is designed to be used when developing programs when use of the AP-120B is impractical or impossible due to production schedules. APSIM emulates all hardware and timing characteristics of the AP-120B as well as performing the mathematical routines as closely as possible to the way the AP-120B would perform them [32]. APSIM requires 32K words of memory in the host computer [1].

f. APDEBUG

APDEBUG is the AP-120B interactive debugger program to be used for dynamic debugging of AP-120B applications programs at run time. Changes can be made when the problem is identified and APDEBUG will call the APLINK and APAL routines to insert the new object module then continue with program development. APDEBUG can work in conjunction with the simulator or the actual hardware [6].

g. Testing Software

There are three software modules available to completely test the AP-120B hardware operations.

APTEST is the AP-120B path tester. This software exercises the panel, DMA interface, internal registers and memory to check for proper operation.

APPATH tests the internal data paths of the AP-120B and returns diagnostics upon finding any errors.

Forward/Inverse Fast Fourier Transform Test (FIFFT) verifies correct operation of the AP-120B's arithmetic units by performing Fast Fourier Transforms and inverses them comparing results with standard answers [32].

These packages can be used to help insure proper operation of the AP-120B before development or actual operation and also help with the hardware fault locating effort during system maintenance.

2. Programming Language

The Math Library of AP functions can be called by the host Assembly Language, Fortran or the AP Assembly Language [36]. However to write a custom library function, AP Assembly Language must be used and the cross-assembler will translate it into an executable routine.

Investigating the programming language is not important here except to say that it is similiar in

characteristics to other assembly languages. There are sufficient commands available to write a program to properly control AP-120B execution in an efficient manner. Bit testing, conditional branching, flag setting and arithmetic instructions all are part of the instruction repertoire which allows varied applications programs to be written.

3. Page Select Option

The AP-120B can alternatively be equipped with a Page Select Option. This provides the ability to address one megaword of main memory in the AP-120B by using host main memory and virtual memory techniques. Each page can be up to 64K words long (full Main Data Memory size but each page must be at least 8K) and 16 pages are available. The Page Select Option increases the ability for the AP-120B to work on larger transforms, but due to paging overhead, it may not increase the throughput rate due to increased host involvement.

This option modifies the AP Direct Memory Address Register (APDMA) located in the DMA Control section of the interface by extending it from 16 to 20 bits therefore 2**20 addressing capability (approximately one megaword). This virtual memory ability is called the AP Memory Address Extension (APMAE) and new addresses can only be loaded by the host. Since the host will control all paging operations, the AP-120B commands will not change inasmuch as it will only recognize 64K word locations [9].

4. Programmable I/O Procesor

The Programmable I/O Processor (PIOP) is a micro-codable micro-processor which acts like a high speed channel program controllinq an input/output port. It is capable of transferring data at a six megahertz burst rate or at a three megahertz sustained operation rate (assuming 167 nanosecond Main Data Memory). The PIOP can be used with up to eight external devices (like A/D converters or mass storage devices) thereby actinq as an I/O bus controller.

The PIOP interfaces directly with the DMA Controller in the interface unit. It has a 38-bit instruction word, a 20-bit arithmetic logical unit and is capabale of addressing to one megaword of memory making it compatible with the Page Select Option. Communication with the AP-120B is accomplished via one of eight flags and four interrupts. The micro code supports subroutines and has the logic to perform jumps within its own code.

The PIOP must handle all handshaking and timing considerations with both the external devices and the host program to insure data integrity. This can be complicated at times so a Programmable I/O Channel (PIOC) is also available which decreases flexibility but eases the programming burden [33].

Neither the PIOP nor PIOC provides a method of connecting two AP-120B's together in series without host

intervention which tends to limit some of the possible applications of the AP-120B.

C. PROGRAMMING, OPERATION AND EXECUTION

The AP-120B can utilize the parallel operation capability of the adder, multiplier and data transfers to increase execution of the program and throughput on large data arrays. These parallel operations must be controlled so that optimum execution speed can be realized without causing interlock or lockout. Lockout could eventually lead to a program stoppage [1]. Since most scientific data can best be structured into an array form, the array processor is able to work on it quickly and efficiently in its natural state where a general purpose computer must, in most cases, restructure it [36].

Before the AP-120B can work on data, the data must first be transferred from its memory locations in the host to Main Data Memory in the array processor (or moved to Main Data Memory from an external device via the PIOP. That situation will not be dealt with here since the PIOP is programmable and therefore path and data options associated with it are many.). The data is transferred via the interface with the use of the APPUT(HOST, AP,N,TYPE) command (Put Data into the AP-120B). As with arguments of other AP-120B CALL statements, HOST AP, N and TYPE need not be explicitly stated but can be expressions, integers or variables.

The host and AP-120B must be synchronized in their operations so computations can not go on while data is still being transferred to memory. APWD (Wait on Data) causes the host to wait until data transfer is completed before it resumes executing the program. APWR (Wait on Running) causes the host to wait until the AP-120B is completed with one command before another is sent over. APWAIT is a combination of APWD and APWR. One difficulty encountered using these commands is that the host to monitor the progress of the execution if polling is used to determine APWD, APWR or APWAIT completion or the AP-120B must wait if priority interrupts are used, which increases the time necessary to complete the program.

Some of the overhead of the host can be eliminated by not using the AP Wait on Running (APWR), AP Wait on Data (APWD) or AP Wait (APWAIT) commands. This technique may speed up program execution and should only be used when it is absolutely necessary and when there is no chance that the results will be processed before they are actually present in the AP-120B Main Data Memory. Floating Point Systems suggests that the program first be written and executed with the APWR, APWD and APWAIT commands present and the results gotten. Then removing a few of those instructions at a time, the results can be checked to see if they match the original results. This only works for specific applications and does not conform to modern programming practices. It is also extremely dangerous since it does not allow for speed

fluctuations due to temperature variations.

When processing is complete, the data can be transferred back to the host via the APGET() command which operates in the same manner as the APPUT.

The application program resides in the host memory and the host executes this program. The host will determine which routines must be passed to the AP-120B and if the data necessary is present in the array processor. When a routine is called, the host will jump to it and execute it but if the routine called is part of the math library (whether from APMATH or a user written math routine), the host first jumps to APEX. APEX then loads the 64-bit instructions into the AP-120B Program Source Memory, calculates the remaining space available in the Program Source Memory, updates the PS location table, loads the parameters and initiates the execution. If the same routine is called again immediately, it will not be reloaded since it is already present but only the new parameters will be loaded. If a different routine is called, APEX will first check the PS location table to see if there is enough unused space available to load it without destroying any routines currently residing in Program Storage. If not enough space is available, the last-written program will be overwritten with the newly called routine (Last In First Out (LIFO)).

The overhead required for each math library routine called is between 100 and 1000 microseconds. One hundred

microseconds is the minimum time required to check the table and move parameters. This minimum time is required for every call, even in looping operations. During this period, the host must be available to the AP-120B which would cause unnecessary host overhead. While the AP-120B is executing any specific routine, the host can be freed to do other tasks and treat the AP-120B as a peripheral device. The host can either be interrupted or can use polling techniques to determine if the array processor requires assistance. In either case, the programmer must be aware of when a break occurs so he can insure that the proper sequence of routines is used to allow the host to perform other operations and not be burdened by many AP-120B services.

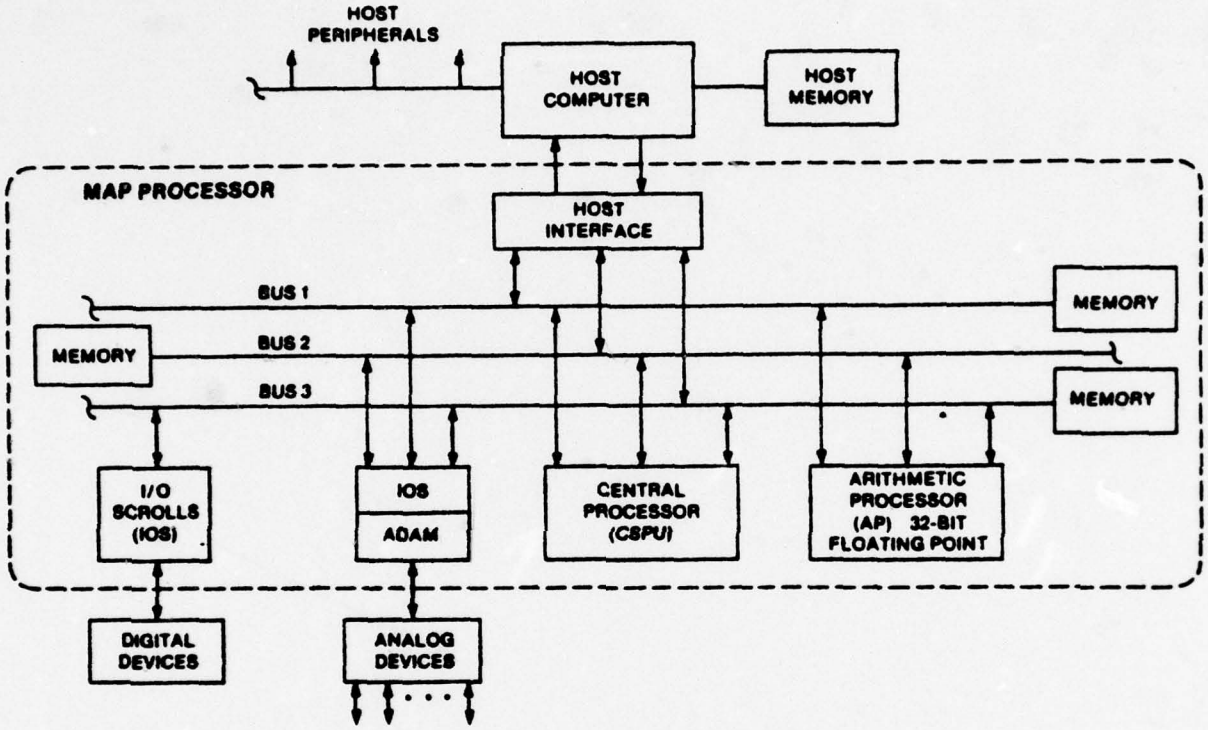
Several ways to increase available free time in the host are to transfer more than one vector with each APPUT or APGET command, use optimum AP-120B library calls to perform given operations (it is the programmers responsibility to determine which AP routines are best for each situation) and overlap host and AP-120B operations whenever possible. Since every call of a routine requires host intervention, several routines can be combined into one by writing a special macro combining those routines, which will effectively eliminate some host overhead by using only one "call" statememt. (But these macros must be small due to limited AP-120B program memory.) Since host overhead varies between 100 and 1000 microseconds, with the higher value being due to the maximum amount of data and program

transfer, some overhead can be eliminated by loading the most used routines first, since overwrite is accomplished by LIFO. APEx must also be a part of the interrupt priority scheme of the host (interrupt or polling); therefore, by having the AP-120B at a high priority, the overall wait time of the system due to interrupt wait can be minimized [8].

V. MAP-300

The MAP-300 (Macro Array Processor) (fig 13) is manufactured by CSP Incorporated, Burlington, Massechusetts. The basic structure consists of three independent busses, an executive routine, two parallel arithmetic units, an addresser and an input/output handler, each having its own clock and operating in a parallel asychronous fashion. The basic logic units are the Central System Processor Unit (CSPU), the Arithmetic Proceessor (AP) (consisting of the Arithmetic Processing Unit (APU) and the Addresser Processor Section (APS)), the Host Interface Scroll (HIS) and an optional Input/ Output Scroll (IOS). All except the CSPU use micro-coded routines stored in their own small memories and communicate with each other via flags set in registers. (The CSPU stores its micro coded routines in main MAP memory.) The Host Interface Module (HIM) section of the HIS, the IOS and the CSPU are built around a standard Intel 3002 bit slice micro processor.

The representation of MAP-300 numbers is usually a 32-bit floating-point format with a one-bit sign, a seven bit exponent (giving a range of 16 ** -64 to 16 ** 63 biased by 64 therefore 0 to 127 are the actual numbers stored) and a 24 bit mantissa allowing a total range of 10 ** -77 to 10 ** 76. Sixteen-bit floating-point and 16-bit fixed-point numbers are also available. MAP-300 main memory is



MAP Configuration

Figure 13

addressable in either 32-bit full-words or 16-bit half-words but eight-bit bytes can be accessed by packing pairs into a 16-bit half-word [18]. SNAP-II commands like VFIX8 assume this packing exists [34]. The ability to address in half-words and/or bytes is important as it may increase the efficiency of the program and array processor, allowing operations to be performed which may not have otherwise fit in a word-only addressable memory.

Although the MAP-300 is asychronous, the advertised average CSPU cycle time is approximately 70-nanoseconds with about 500-nanoseconds required for a memory read/write operation when using 500-nanosecond MOS memory (125-nanoseconds using bipolar). Full-word operands and results starting on an odd address boundary, however, require about two 500-nanosecond memory cycles. A pseudo-operation can be used to insure even-boundary locations exist [18].

The MAP-300 is capable of operating in temperatures from 0 to 50 degrees centigrade at 10 to 90 percent humidity. The power requirements are either 115 VAC or 230 VAC single phase plus or minus ten percent at 47 to 63 hertz. The weight is approximately 80 pounds.

The MAP relies heavily on internal parallel processing to increase throughput and limit wait time. The MAP-300 stores the executive and array routines in its own memory (as opposed to storing it in the host memory). With the use

of function lists and statements like "MPWHL" (MAP version of the "DO WHILE"), the MAP can operate independently of the host after initial loading of the program [19]. With the three bus structure, the MAP theoretically can simultaneously input into one memory, output from the second while doing computations on the third and never utilize the host except for initialization.

The MAP has a separate instruction set for the Central System Processor Unit (CSPU), Arithmetic Processor Unit (APU), Addresser Processer Section (APS), and Host Interface Scroll (HIS). Inasmuch as these processors work independently, the instruction sets are not as complicated as may have been necessary if operation was controlled totally from a central site. The total number of instructions per second attainable by the MAP-300 is data dependent. Whenever all steps necessary to perform the operation are completed, as witnessed by properly setting the correct flags in pseudo-memory (to be discussed later), the operation will perform to completion. While the addition/multiplication operaton is being carried out in the APU, preparation for the next word (half-word) of information can be conducted in the unaffected processors. System flags are used to communicate between the processors. These flags include General Purpose flags available to the programmer for general system communication, Control flags to control processor modes and operation sequencing, Status flags to indicate processor status and Hardware

Configuration flags [18].

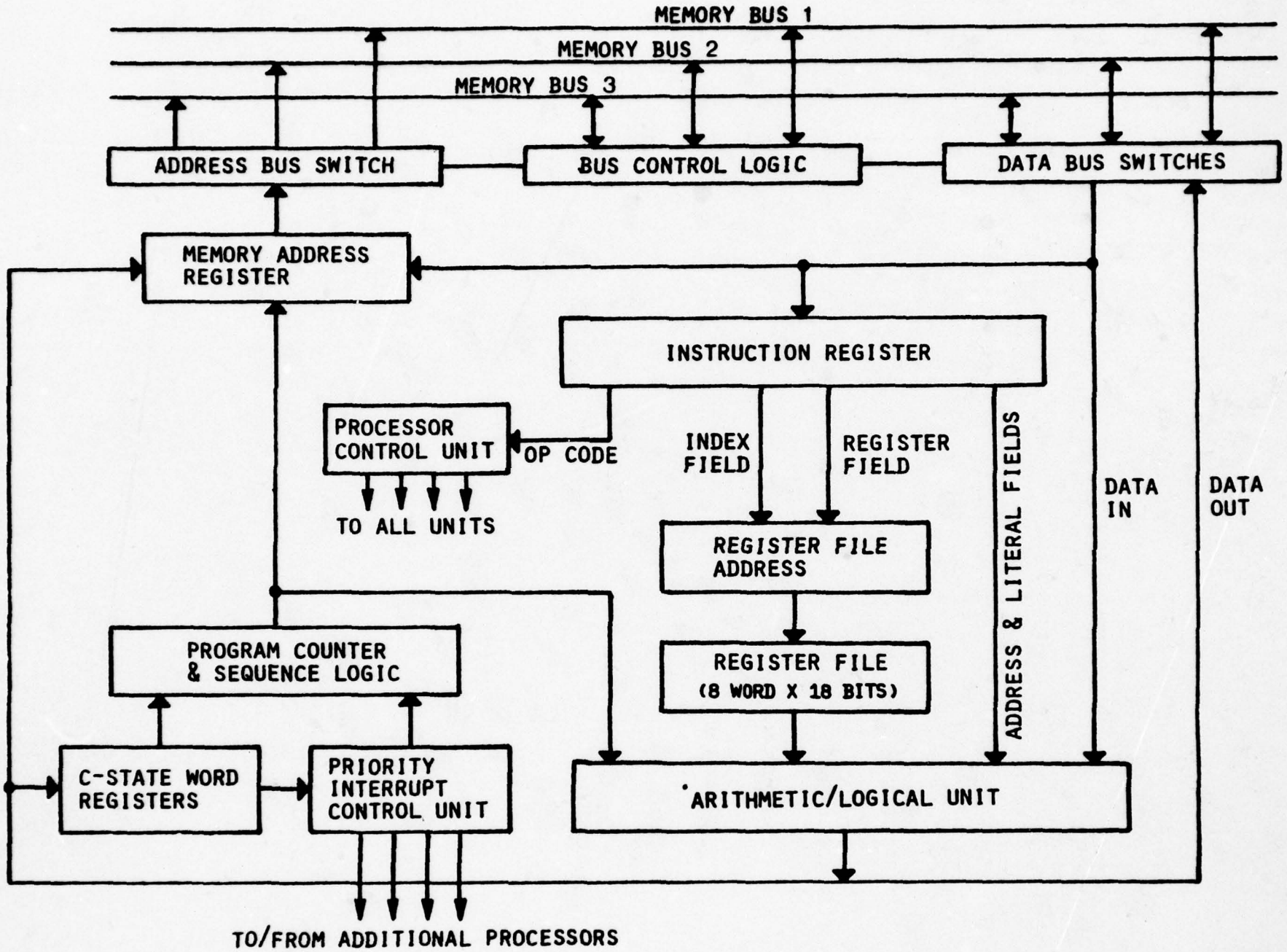
The MAP-300 system installed for evaluation consisted of: the MAP-300 processor, interface with the PDP-11 computer utilizing the RSX-11M operating system, 24K words of 500-nanosecond MOS master memory (8K for each memory), power panel, expansion chassis, installation, I/O driver, SNAP-II algorithm library, cross assembler, simulator and loader. The price of the system was $44,500 [27].

A. CHARACTERISTICS AND HARDWARE

1. CSPU

The Central Processor Unit (CSPU) (fig 14) is the "Command Central" of the MAP-300 array processor. The CSPU responds to commands from the host, transfers data to and from the host, assists the APS in address calculations and loads the program memories of the Arithmetic Processor and Host Interface Module. The CSPU performs the functions of a front-end micro computer to control the actions of the system.

The CSPU has a fast, fixed-point arithmetic unit for address calculations, an instruction register, an eight register accumulator file and a priority interrupt network. It has access to the three main memories via the memory busses and supplies the other MAP processors with the program instructions they need from main memory. Reentrant



CSPU Block Diagram

Figure 14

subroutines and multi-level indirect addressing are recognized by the CSPU. It has no I/O capability but instead instructs the Host Interface Scroll (or I/O Scroll) to perform input or output operations to or from the host (or external devices). The CSPU will never halt but will always be in the WAIT state after its instruction sequence is completed.

An important register in the CSPU is the Control Status Register or C-State Word (CSW). It is a 32-bit register containing the status of prior operations, the program counter as well as the source and destination locations for block memory transfers. Fields of the register can be combined to give hardware condition codes for use in conditional operations, branches, jumps or executes. The CSW also stipulates on which bus instructions or data are present and controls the interrupt responses for other units.

The CSPU is the only processor able to be interrupted in the MAP (other processors can either Halt or Wait) and contains a 64 level interrupt priority system with one interrupt device per level and three lines per device (192 possible combinations). The CSPU may only be interrupted between instructions. It will also nest and queue lower priority interrupts if a higher priority interrupt is received during the servicing of a lower priority interrupt. These interrupts are detected by

polling and levels are polled only if they are above the current interrupt level. Lower level interrupts will continue to exist but will not be recognized until the higher priority interrupts are serviced.

The CSPU contains no memory but uses main memory to store its instructions. When fetched, these instructions are stored in the instruction register until execution. The CSPU may also address a pseudo-memory location called System Flag Register (SYSFLG) which is the primary inter-processor communication system. By testing the bits of SYSFLG, the CSPU can sense the status of any of the other processors. (Pseudo-Memory refers to memory physically located within the sub-processors but which appear on the bus as a memory address similiar to the PDP-11/34/45/55/60/70.) [18].

2. Arithmetic Processor

The Arithmetic Processor consists of two components, the Arithmetic Processor Unit (APU) and the Addresser Processor Section (APS).

a. APU

The Arithmetic Processor Unit (APU) (fig 15) is responsible for the computation required in array processing and executes programs relatively independent of the other MAP processors, operating under the general control of the CSPU . The APU consists of two adders, two multipliers (the main distinction between the MAP-300 and the MAP-100 or

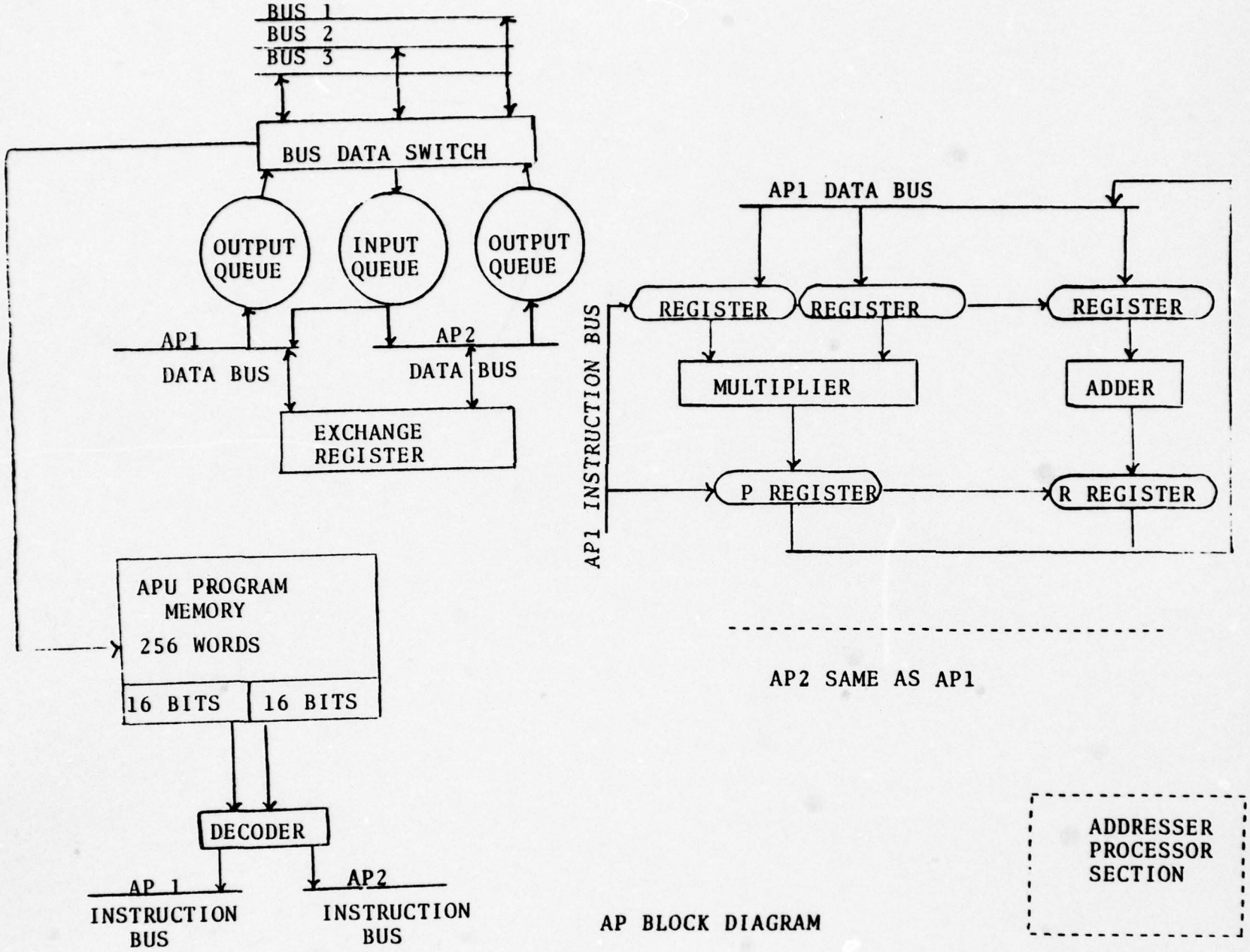


Figure 15

AP BLOCK DIAGRAM

MAP-200 is that the former contains two adders and multipliers while the others contain only one each), 34 various registers and three First-In-First-Out (FIFO) buffers for input and output storage. The two adders and two multipliers permit parallel processing of data to increase throughput. APU programs are stored in main MAP memory and are sequentially block-transferred to the APU program memory under control of the CSPU.

The main units of the APU are the arithmetic processors (AP1 and AP2). Each arithmetic processor consists of an adder and multiplier that may operate simultaneously and independently of each other. Each adder is fed by eight registers and each multiplier by four multiplicand registers and four multiplier registers. The results of the adder are routed to the result register R and the multiplier loads the product register P. To transfer data between the separate arithmetic processors, an exchange register is provided.

APU memory consists of two 256-word 16-bit sids-by-side memories. The memory is initially loaded by the CSPU from MAP memory and the APU is then put into the run state. Instructions are sequentially decoded in the APU to perform the specified algorithm. The instructions are 16-bits for each board (AP1 and AP2) and are executed in parallel. They can perform addition, multiplication, transfer of data and the setting of flags. These

instructions are decoded and the operation started as soon as all necessary conditions are met. Immediately, the next instruction is retrieved and decoded and attempts to be executed. If either the P/R register is involved in a multiplication/addition operation which has not yet been completed, the Input Queue(IQ) is empty or the Output Queue (OQ) is full, the APU will go into a "wait" state. It will remain in this "wait" state until the multiplication/addition instruction is completed or the other conditions are satisfied. There is a problem that can exist due to the sids-by-side 16-bit memories used for program storage. Since there is only one program counter and the AP1 and AP2 processors work in parallel the sids-by-side memory acts as two halves of a 32-bit instruction register. Therefore if one board (AP1 or AP2) is forced to wait, the other must also wait since the next instruction may not be retrieved until the program counter can be incremented.

The Input Queue is a four-deep FIFO buffer which services both AP1 and AP2. To get the next input data field, the IQ must be advanced before the data is transferred. If both boards request data without advancing the queue, they will receive the same data, which may be good for certain applications. If they both simultaneously try to advance the IQ, it will advance only once and give an AP1 priority, then advance the second time after the transfer has been completed to give data to AP2.

There are two Output Queues each of which is a four-deep FIFO buffer. These queues allow maximum capacity of the adder and multiplier to be utilized, since it is less likely that the processor will have to wait for either buffer to have a vacancy due to a busy bus system. If both processors try to act on any single OQ, processor AP1 will be given the priority.

A typical multiplication takes approximately six cycles (420-nanoseconds) and a typical add takes about three cycles (210-nanoseconds). Therefore, to increase throughput, "hiding" adds, moves, etc. behind multiplies will accomplish operations in the time it takes to do the multiply alone. The most efficient method to program the MAP-300 is to treat successive sample sets in alternate processors; this effectively produces a multiply every 210-nanoseconds. Since there is one input queue, this method allows both to have access to the same information (by not incrementing the queue) and also gives a greater chance to use hiding effectively.

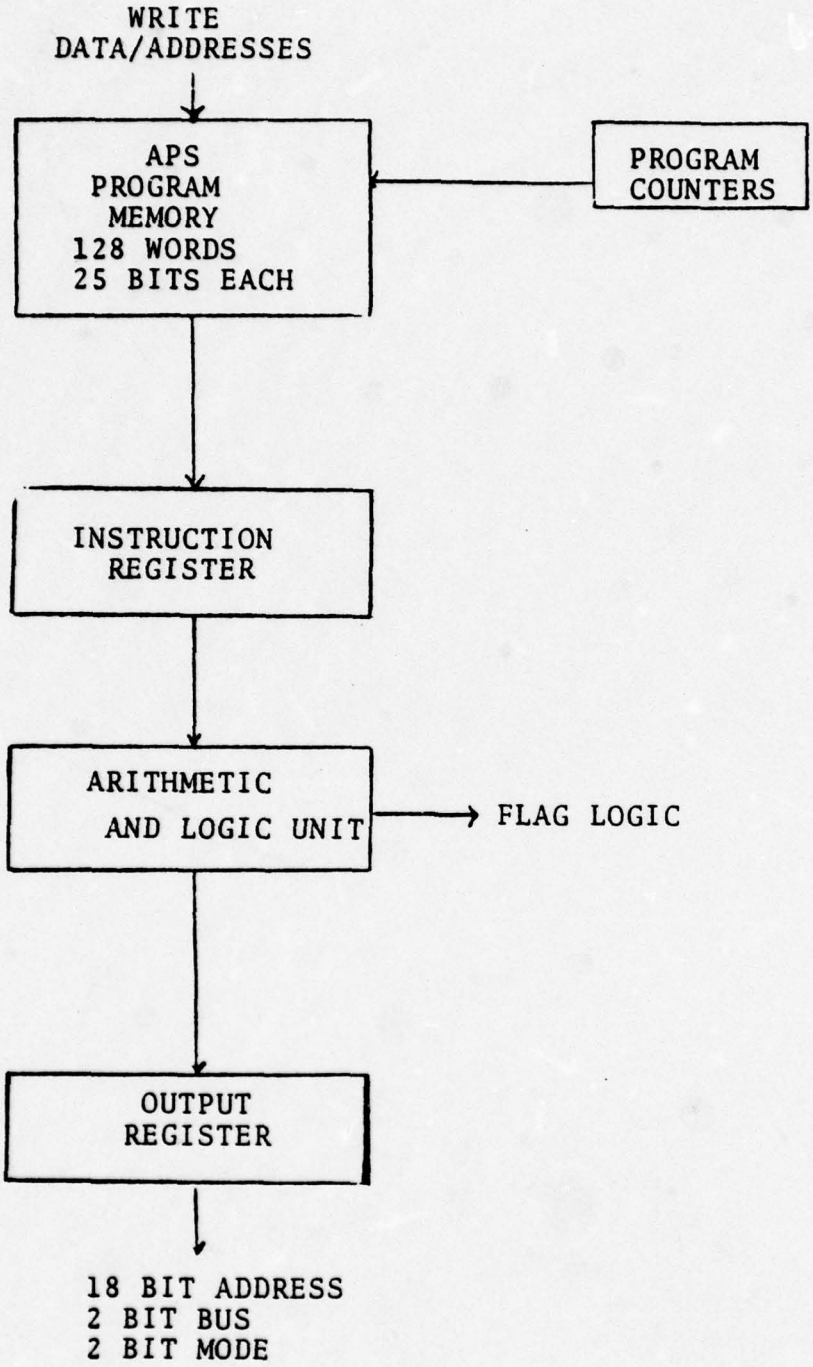
The APU can usually operate in two modes. Mode One, the normalized mode, can either use normalized or unnormalized floating-point numbers as input with the results being a normalized floating-point number. Using unnormalized floating-point numbers as input can lead to precision loss since the normalization process will shift the mantissa to the left (values less than .1) or to the

right (values greater than 1.0). The vacancies created by these shifts will be filled with zeros, which, after computation, could possibly produce an unusual truncation. The unnormalized mode will accept unnormalized numbers as input and will return unnormalized numbers as output [18].

b. APS

The Addresser Processor Section (APS) (fig 16) computes both the address in MAP memory for the location of input data words to be processed by the APU and the MAP memory addresses for the output from the APU. It operates independently of other processors, within status and control flag constraints of SYSFLG. The APS contains a 128-word 25-bit memory, four program counters (two for read and two for write), eight address buffers (to be used as inputs to the adder), four First-In-First-Out (FIFO) buffers, an arithmetic logic unit (adder), and associated logic and control units.

The APS programs are stored in MAP main memory and are loaded by the CSPU. Certain absolute address locations must be known to a APS program at run time which are not available during program writing. The assembler computes them at assembly time and the CSPU inserts them into the proper location during this program transfer. The CSPU then initiates APS operation by setting the proper flags. The APS may be loaded with new information by the CSPU during run time by cycle stealing, thereby not causing



ADDRESSER PROCESSOR SECTION

Figure 16

the APU to slow and wait for a value in the IQ or a space in the OQ. Because the instructions in MAP memory are 32-bits long and the APS instruction is only 25-bits long, the seven bits left over are used to store the APS memory address for that instruction. This allows the CSPU to increase throughput by immediately installing the instruction into the correct location in a pre-computed order.

The adder computes addresses dependent on prior computational results, literals or specified increments. All address addition and subtraction is considered to be modulo 2 ** 17 so that only positive addresses in that range will be computed. Results are queued in either the Read Address FIFO (RAF) or Write Address FIFO (WAF). Along with the address is a code to delineate whether the address is full-word, half-word or byte (pair of bytes in a 16-bit half word address) and if it is a eight-bit fixed-point number, 16-bit fixed-point number, 16 bit floating-point number or a 32-bit floating-point number.

The distinctive feature of the APS is that there are four program counters (P0, P1, P2 and P3). These allow four separate programs to be stored in the APS and executed in an interleaved manner. Sequencing of these programs is controlled by the status of the WAF and RAF in conjunction with the APS instructions. These program counters also provide a looping ability allowing the APS to work with the Host Interface Scroll or I/O Scrolls to keep data flowing.

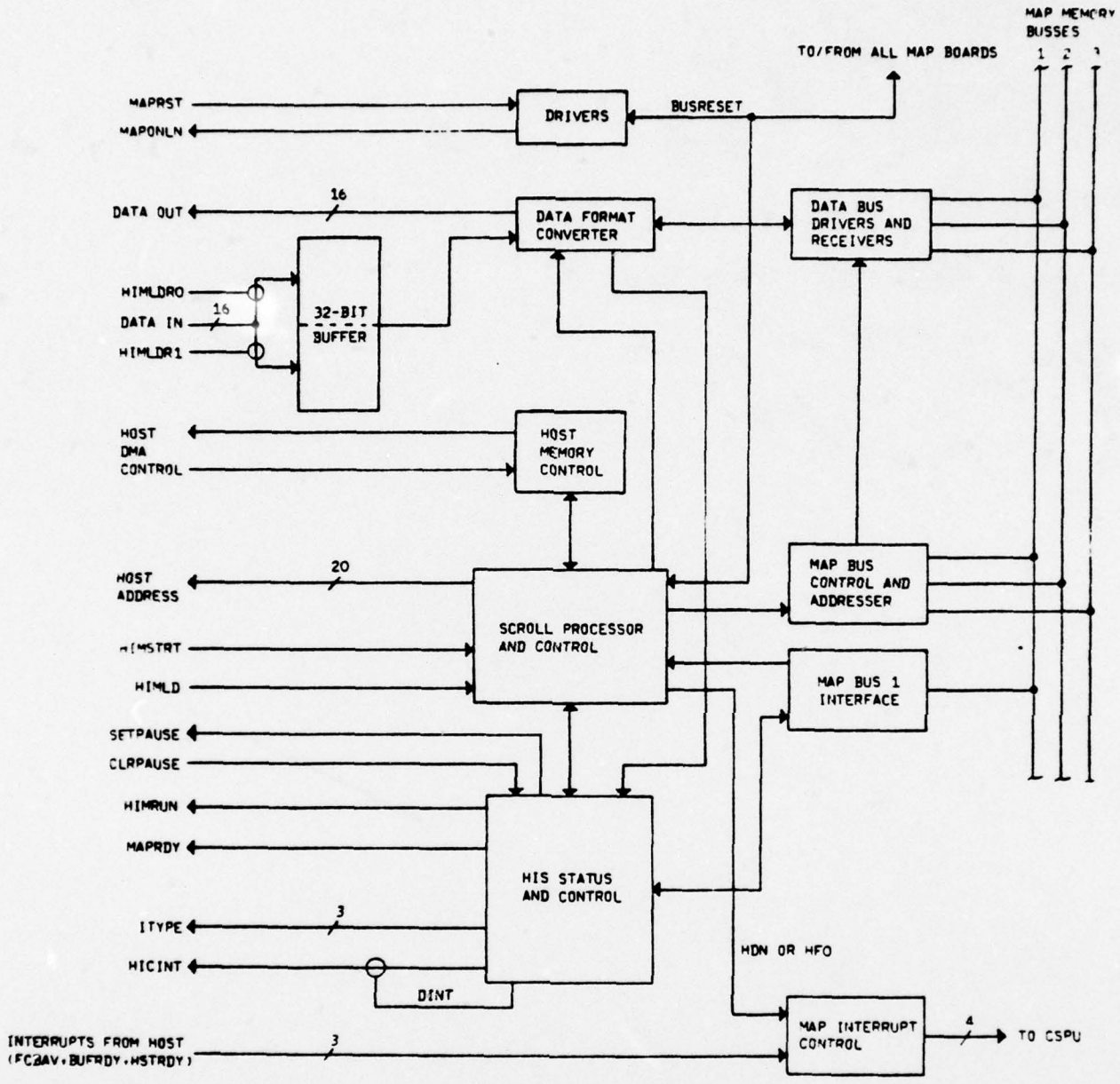
After one memory has been processed and reloaded, the APS need not be reinitiated but can continue operation on the new data by this looping feature [18].

3. Host Interface Scroll

The Host Interface Scroll (HIS) consists of two subsections, the Host Interface Module (HIM) (fig 17) which is located in the MAP-300 and the Host Interface Controller (HIC) which is located in the host memory. The host Interface Module transfers MAP programs, unprocessed data, host status and Host Interface Controller commands from the host to the MAP. Processed data, MAP status and processing commands are also transferred from the MAP to the host via the HIM. A programmable scroll processor is provided for computing MAP and host memory locations during a Direct Memory Access (DMA) operation. Other pertinent devices include a memory-bus interface, controllers for host memory, format conversion hardware, status and control logic along with interrupt logic.

The HIC controls the handshaking necessary between the host and the MAP. The handshaking consists of interrupt logic from MAP to host and logic necessary for controlling the transfer of data with either Direct Input/Output (DIO) facility or DMA transfer [18].

The host generally interrupts the MAP to initiate program sequencing. However, when the MAP is completed, it



Simplified HIM Block Diagram

Figure 17

will initiate communication (interrupt) with the host for further work. When the interrupt is acknowledged by the host, more data or programs are sent to the MAP depending on the flags. (If all MAP processors are in a loop operating on data supplied from external devices and delivered to external devices via I/O Scrolls, the host will not be interrupted unless there is an error. This frees the host to do any other unrelated processing necessary.) The maximum response time to initiate an interrupt is 150 microseconds for the HIM and 250 microseconds for a user CALL routine [35].

4. Memory

Main memory in the MAP-300 consists of three independent busses each having the capability of 256K words of 500-nanosecond MOS memory or 64K words of bipolar memory. Memory types may not be intermixed on any given bus but each bus may have a different type from another bus. Memory can also be either master or slave, master memory being used to control program execution, arbitrate and observe system protocol while slave memory stores the data. Each memory bus containing memory is required to have at least one master memory module (available in either 4K or 8K blocks for MOS or 1K, 2K, or 4K blocks for bipolar).

Access to each memory is via a common bus having 11 ports and two priority levels. Three ports are reserved to

be used with the absolute priority scheme leaving eight ports with a sequential round-robin (polite) priority scheme. Absolute priority is the highest priority and is intended to be used with high speed minimally-buffered devices such as disc units or tape units where loss of data may result. Sequential round-robin priority handling is used for slower buffered devices and is a round-robin (circular) queue which is checked each memory cycle. The device first in the queue will get the next memory cycle. Scanning for the next queued device will commence immediately upon the previous device starting tranfer. When the next memory cycle occurs the new device will be known keeping overhead minimal. Of these 11 ports, the HIS and CSPU each have one dedicated port and the AP has two dedicated ports on each bus with seven ports remaining for the IOS and other uses.

Psuedo-memory (alluded to earlier) is the upper 4K words on Bus 1 containing addresses of certain registers used for status and control. These registers are located in the sub-processors but appear as addresses on the memory bus. Any sub-processor may alter the contents of these locations so it is important that the programmer not try to overwrite these addresses with programs or data [18].

B. SOFTWARE SUPPORT

As with the AP-120B, there are software routines to aid in program development and execution.

1. Executive and Associated Routines

a. Assembler

The MAP-300 assembler, written in ANSI Fortran IV, takes a source program written for either the CSPU, APU, APS, HIS or IOS and creates an executable object module. A listing file and errors file can also be created. Editing and updating can be accomplished from the last source file by changing and assembling only the incorrect line (or lines) of code, thereby avoiding the reassembling of the entire program [18]. The assembler will also allow change of the HIM memory to enable it to handle necessary buffering.

b. Simulator

The MAP Simulator Program simulates model 200 and model 300 processors by executing MAP object code. The simulator permits the programmer to develop or debug software off-line so as not to disturb production schedules.

The MAP Simulator Program has the capability of simulating the operation of the APU, APS, CSPU, Memory and the interrupt handler. It has not been updated to handle certian new commands and flags (listed in the front of ref[25]) nor does it have the ability to simulate the APU

test mode. Memory size and type can be specified either in the initial loading of the simulator or while running to tailor it for current or proposed configurations.

When used as a debugging aid, the MAP Simulator Program allows the operator to: install breakpoints and execute macro instructions at these breakpoints; detect program errors and execute macro instructions after their discovery; examine register contents; run programs from different processors (APU, CSPU, etc.) independently; and, patch loaded programs. Input/output may be obtained from a terminal, printer, tape(magnetic or paper), cards or cassette. A batch mode is also available. Actual program timing can be estimated by installing breakpoints and individually timing small sections of code [25].

c. Loader

The MAP Loader is a Fortran program which accepts object code produced by the Assembler and create blocks of binary code in MAP machine language. This code is transmitted to the MAP memory via the MAP driver through the Host Interface Scroll. Errors in transmission are detectable since check-sum digits are transmitted to the MAP along with the blocks of code. The Merge operation creates and updates the tables and addresses necessary if the loaded module is to be used with the SNAP-II executive [22].

d. Debug Package

The MAP-300 diagnostic package is designed to verify hardware operations and isolate any malfunction, to a specific card. One module is resident in the host while another, which contains the test modules and test programs necessary to determine proper system operation of the CSPU and other sub-processors, is present in the MAP. This software can run interactively or under batch processing [18].

The MAP-300 LOOK program permits the programmer to examine MAP memory (or pseudo-memory) from any computer capable of operating under ANSI Fortran IV. This is also an interactive routine and provides the ability to "patch" coded program segments or enter entire machine language programs. The programs or segments can then be stepped through to examine the results closely [20].

2. SNAP-II

Systematic Notation for Array Processing Version II or SNAP-II is a single-command high-level macro-type language used to program the MAP-300 array processor. The SNAP-II package consists of a Host Support Module, Host/MAP driver module, SNAP-II Executive, SNAP-II Function Modules and an Installation test and Acceptance test Module [18].

The SNAP-II executive permits the user to define buffer size, and the structure and location of programs in

MAP memory. The executive also structures the routines to operate at maximum speed by insuring that the maximum possible parallelism exists between sub-processors (for CSPI written functions), thereby accentuating "hiding". The SNAP-II subroutines are written in ANSI Fortran and passed to the MAP via Function Control Blocks (FCB). The MAP Driver, which is located in the host, directs the loading and operation of the programs. (In a loop or "Map While" condition the driver need only load and initiate the sequence then return control to the host operating system.)

SNAP-II allows the programmer to build his own function lists with the Fortran type statement "Map Begin Function List" (MPBFL()) which permits the host to remain as free as possible from the operation of the MAP. Two-dimensional arrays are demultiplexed by SNAP-II thereby increasing speed of execution in the processor by not having to compute two-dimensional address structures. SNAP-II functions are callable from either ANSI Fortran or Host assembly language programs and are able to operate on both real and complex data [15].

3. Programming Language

If SNAP-II functions are not specific enough to satisfy the programmer's needs or if they do not exist in the SNAP-II library, new routines may be written in an assembler type language. The CSPU, APU, APS and HIS each have their own instructions to optimize each sub-processor's

capabilities.

The CSPU instructions are broken into 10 groups which have the ability to perform all the functions that a general purpose computer is normally visualized as performing. They include: generic (performs interrupt system coding and looping); single register; move; logical; push and pop; hop and jump (a hop is within 256 half-word locations and a jump can be to any new location); skip and bit manipulation; compare; and maintenance and test console instructions. The APU can perform: two-argument adder; single argument adder (like approximate reciprocal instructions); multiply; data transfer; jump and call; and control operation instructions. The APS performs: load; address increment; register arithmetic and control type instructions. The HIS recognizes: single register; logical register; arithmetic register; literal and control instruction types [18].

Since each sub-processor is designed to perform a special operation and can be programmed to optimize that design, the overall performance of the system is increased. All processors perform in parallel and stay in "sync" by the use of flags. A sub-processor will wait until the proper flag is set before continuing, thereby insuring integrity. The waiting also relieves the programmer of "counting cycles" with No Operation (NOP) instructions which could possibly cause lost data. The drawback is that he does have

an increased complexity by insuring that proper flags are set at the proper time [16]. Most of these encumbrances are eliminated by the executive however. Flags are available in pseudo memory and are easily tested. The complexity issue is minimal since for most applications only APU and APS routines need be written. Only under special circumstances is a CSPU or HIS routine required.

Pseudo-operations are also available to ease the programming burden. They perform such tasks as naming character strings, insuring that information is placed into memory on a word boundary, generating constants and making a test Control Status Word (CSW).

4. I/O Scrolls

The I/O Scrolls (IOS) control block-transfers to or from external peripheral devices (including other MAP's) without interferring with the MAP-300 processing cycle by using a sub-processor which can be pre-programmed. The IOS contains three functional elements: protocol logic necessary to interface the external device directly to the MAP-300 memory busses; a programmable processor to compute MAP addresses and issue control signals; and, the transfer logic necessary to interface with peripheral devices.

There are five basic IOS models. IOS1, also known as the maintenance and test console, is capable of transferring eight-bit single words to MAP bus number one at

a 5 KHZ rate. IOS2 has two transfer rate options and two word size options available. Word size option one utilizes the block-transfer of 8 or 16-bit words to any of the three MAP busses while option two uses either 16 or 32-bit words. Transfer rate option one conveys information at a 1 MHZ rate as compared to the 2.5 MHZ rate of option two. Either transfer rate option may be combined with either word size option; however, only one combination is available at a time since they are hard-wired. Under program control, IOS3 can transfer either 16 or 32-bit words to any of the three busses at a 750 KHZ sustained rate. IOS3 can also perform format conversion, monitor data with a basic operation similiar to the HIM and support indirect addressing. IOS4 is a high speed (up to 40 MHZ) scroll, allowing block transfers only of 8, 16, 32 or 64-bit words to any bus (64-bit words must be transferred simultaneously to bus 2 and bus 3). IOS4 also allows packing and buffering of data [18]. IOS5 is a direct memory-to-memory bus-connect option for direct data transfer between user devices and the MAP-300. The module requires no software (and will not support software). Its operation is controlled by hardware and three interrupt request lines [21].

a. Analog Data Acquisition Module

The Analog Data Acquisition Module model 5120 (ADAM-5120) is a programmable analog interface capable of accepting from 2 to 16 channels of analog information. This

information is then digitized to 12-bit resolution at a 270 KHZ throughput rate for the 16-channel case (125 KHZ for single channel). As with the I/O Scrolls, the A/D operation may take place simultaneously with the MAP-300 processing. The ADAM is functionally equivalant to the IOS2 with only added analog-to-digital circuitry. This allows the ADAM to be SNAP-II compatible.

The operation of the ADAM is carried out via a set of up to 16 sample-and-hold units which then make their signals available to a 16:1 multiplexer. Each channel of the multiplexer is the consecutively sampled by the A/D converter which outputs either a 16-bit sign-magnitude or 16-bit floating-point number. Performance accuracy is specified a 0.2 percent of full-scale resolution [2].

C. PROGRAMMING, OPERATION AND EXECUTION

The MAP-300 can not only utilize parallel operations of the adder and multiplier in the APU, but also the parallel sub-processor operation of the APS, HIS, IOS, APU and CSPU to increase total throughput. The programmer, by breaking the problem into smaller independent programs of addressing, arithmetic, I/O and management, can theoretically more easily program the entire problem than by adherring to internal communication protocol and flags [18]. The respective programs should be easier to write with much of the increase in overhead due to the added handshaking and

protocol requirements being assimulated by the executive. [16].

CSPI recommends that a modified top-down programming technique be used initially by writing the APU routine first to insure the optimum execution speed. Then adding the other necessary routines (generally just the APS routines) to insure the information is present when the APU needs it. The APU should be programmed to treat subsequent sample sets in alternate adder/multiplier modules and arrange data so that as many adds can be "hidden" as possible [18]. By proper execution, sequencing total time can be shortened to equal the time to multiply only, with all other operations "hidden" under these multiplies. This "hiding" operation becomes easier in the MAP-300 than in the AP-120B since cycles need not be counted and NOP's need not be inserted for unused cycles due to flags being set to signal the availability of resources [16]. The programmer must be aware that the timing is not absolute, therefore the executive will tightly control synchronization by flags to insure one adder/multiplier does not get ahead of the other.

The programs are initially loaded from the host to the MAP via the operating system interface and driver. The MAPDVR.MAC routine makes the standard interface through the operating systm and MPDRV.MAC makes the MAP appear as a standard RSX-11M device to the computer. Initial communication from the host to the MAP is done via a four

word Driver Control Block (DCB) [26]. When the Central System Processing Unit is initialized by the host, it will load the other sub-processor programs and commence program execution.

Subsequent MAP commands are sent to the MAP from the HOST via Function Control Blocks (FCB) which require host intervention to send. (Function lists and the MPWHL macro treat multiple FCB's as a single entity). These FCB's transmit host to MAP status, interrupts and functions to perform and can be queued in the HIS buffer. When it is no longer necessary for the host to send or receive a FCB, it can perform other operations [35]. Therefore, with efficient use of the IOS and the possibility of stringing MAPs in series, the host can be free to either perform other tasks or act as a system monitor.

AD-A066 370 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 17/1
EVALUATION OF A SIGNAL PROCESSING TEST BED.(U)
DEC 78 G T VRABEL

UNCLASSIFIED NL

2 OF 2
AD
A066370

END
DATE
FILMED
5-79
DDC

VI. DISCUSSION OF FINDINGS

In the test bed, the PDP-11/34 was chosen to perform the front-end functions which consisted of buffering the data, formatting it and then passing it to the array processor or mass storage device (or from the mass storage device to the array processor). This limited front-end inputting function did not dictate that the computer be large. The choice of the PDP-11/34 computer for this application seems adequate. The PDP-11/04 would normally contain enough speed to handle the necessary operations but may be unsatisfactory since it does not have a resident memory control and protection routine to ease the programmers burden and help insure system integrity, nor does it contain the 2K cache memory to increase speed. A computer larger than the PDP-11/34 may not increase the efficiency of the system although it would increase the cost.

The test bed utilized the PDP-11/70 for the output computer. The output computer would be required to receive information from the array processor, manipulate the data and store it for future display on one or more devices. For this application, the PDP-11/70 seems best for several reasons. The system is much like the 11/34 except that the current maximum memory is 2 megabytes to allow for better utilization of information. There are dedicated paths to

high performance storage devices that would allow more information to be processed per unit of time. To further process arrays for output, there is a 32-bit or a 64-bit floating-point arithmetic unit available. The PDP-11/70 gives large-computer performance and expansion capabilities with the cost and space requirements of smaller units [31]. Using the same manufacturer for the output function as was used for the input function reduces interface problems and contributes to the proficiency of the programmers by increasing overall knowledge of the architecture.

The proposed test bed uses of the 11/34 and 11/70 can be greatly modified by the choice of the array processor. The MAP-300 utilizing an Analog Data Acquistion Module and/or I/O Scroll can eliminate the need for the input functions (including 16 channel analog-to-digital conversion) therefore permitting the 11/70 (or possibly a less costly model) to perform input, output and monitor functions in the test bed. In fact, the 11/70 will probably be large enough and fast enough to facilitate combining all subsystems, except the display subsystem, under one computer. The 11/34 and 11/70 combination should provide for the full range of computers necessary to properly emulate and evaluate just how much computer capability will actually be needed for any specific application.

The question arises as to which is the best array processor for the application. The AP-120B is synchronous,

therefore some may say safer, has a 38-bit word which could mean greater accuracy, more standard library functions (such as vector log base 10 and vector log base e) and a 3500 hour mean time before failure. The MAP-300 is a newer system which, due to the minimal host involvement, three separate busses, I/O Scrolls and the ADAM, can provide greater long run throughput and more flexibility.

For the non real-time environment where simple programming and host involvement can be tolerated, the AP-120B may be a good choice. It can provide facilities to tailor algorithms to specific needs; these facilities are not yet too complex to tax the normal programmer. However, new programs cannot be added directly to the AP math library (APMATH) but must be linked and loaded for every usage as would any application program. This creates an excessive time overhead. Therefore, the AP-120B should be used only where simplicity and ease of use are paramount and utility can be sacrificed.

For applications requiring real-time computations (which the test bed most likely will eventually demand) innovative design, high throughput rates and generally greater flexibility, the MAP-300 provides the answer. The improved performance of both array-processing potential and computer availability is offset by the increased cost of program development if non-library routines must be written. These routines however may be added to the library

effectively reducing overhead. Reference [23] reports that the MAP-300 also complies with MIL-E-16400, MIL-E-5400, MIL-STD-461A, MIL-STD-704B and MIL-STD-1399.

During the installation of the MAP-300 at the Naval Postgraduate School, it was noted that the installation documentation was extremely poor. As of this writing, three weeks were required to install the system. This was due mainly to the poor documentation in the installation package received with the unit. Not only was the package incomplete, but changes to the software were performed that were not changed in the original documentation, nor was an eratta sheet provided.

It is realized that for many companies involved in data processing equipment manufacture, documentation is not a chief concern. However, CSPI seems to have far inferior installation documentation than would reasonably be expected. This situation made it impossible to do a good test of the system operation but allowed only a cursory review.

Even with the evident shortcomings of the documents, theoretically the MAP-300 is far superior to the AP-120B. If CSPI would upgrade their documentation and perform the installation at the site, their sometimes negative public image could be eliminated and confidence in their equipment could be increased. It must be noted however that ref [18] and the publication "Simple Notation For Array Processing,

Version II, Reference Manual", are excellently written. There,fore in the following discussion, the use of the MAP-300 will be assumed. I will now look at each subsystem closely and attempt to determine alternate designs.

The analog subsystem obtains data from one of four sources: time code read/generator, 14-track recorder (Honeywell 96), signal synthesizer (Rockland 5100) and/or a noise generator (HP 3722A). Up to 128 channels of input are amplified, sent through a programmable matrix switch resulting in 32-channel output signals to a programmable 32-channel filter. These analog signals then leave the analog subsystem to be input to the signal processing subsystem.

The AN-5400 analog-to-digital converter performs a 12-bit A/D conversion and is then loaded the Ampex Megastore mass storage device through the PDP-11/34 computer. The output of the array processor will then be sent to the data processing subsystem.

I suggest it may be easier, more flexible and cheaper to input the 32 channels as before to the programmable filter, but then the 32 channels may be better handled by two Analog Data Acquisition Modules directly into the MAP for processing or via an I/O Scroll, model 3, be sent to the PDP-11/70 storage devices for future use. This will eliminate the expense of the A/D converter, Ampex Megastore and the PDP-11/34 but more important, it will be relatively

easy to perform calculations in real-time. Once the MAP-300 is started, it can perform without host intervention until interrupted and with an assumed input of 40 KHZ, the system should not be taxed. The output of the MAP can then be sent directly to the data-processng subsystem. The entire system can also be less complex, affording easier system development.

Assume that a fictional system with a 40 KHZ input requires a FFT and discrete digital filter to be done on the information. The timing of a 1024 real to 516 complex Fourier transform requires 3.0 milliseconds [23] and a 40 KHZ input rate would require 39.1 FFT's per second on the average. This would consume 117.3 milliseconds and assuming a 50 percent overhead yield 175.95 milliseconds to perform the Fourier transform. Discrete filtering would require another 39.1 * (1024 * (2 * 500 nanoseconds + 12 * 70 nanoseconds)) or 73.67 milliseconds. Again assuming 50 percent overhead, 110.51 milliseconds would be necessary for the filtering. The total time consumed by the two functions would be 286.5 milliseconds, leaving 713.5 milliseconds for other work. (Fifty percent overhead is an over-estimation.) Loading data into the MAP-300 would be hidden behind the FFT operation (except for the initial case) and would not contribute to overall execution time.

This would effectively eliminate the entire signal-processing subsystem with th exception of the MAP-300. The

PDP-11/70 computer in the data processing subsystem could control the MAP along with its other intended function of controlling the display subsystem. Any storage necessary for output or any taped input data could be handled by the tapes and disks associated with the 11/70 and execution could be performed on the MAP-300 along with the above calculations. However, for expanded utilization, not specifically addressed, the above use of only one MAP and no PDP-11/34 may have to be modified to accomodate the new requirements if these new requirements are significantly larger.

If after extensive testing the MAP-300 proves to be too costly due to unreliable software, the AP-120B can perform the same functions although at an increased hardware and time cost.

For example, in the AP-120B, to perform the above real to complex FFT, it requires 5.08 milliseconds for the FFT, 0.8 microseconds to rescale and 1.7 microseconds to reformat the result for a total of 5.09 milliseconds per 1024 sample FFT. To this must be added 100 to 1000 microseconds overhead for each of the four call statements: Get data from the AP-120B(APGET), Put data into the AP-120B(APPUT), real to complex FFT(RFFT) and real FFT scale and format(RFFTSC). I will use the arithmetic average of 550 microseconds per call for an added 2.2 milliseconds resulting in a subtotal of 7.29 milliseconds per FFT. APPUT

and APGET have no specific times in ref [8], but according to Floating Point Systems the PDP-11 interface transfer rate is 750 KHZ. This would therefore require approximately 2.67 milliseconds for each 1024 element transfer giving a total of 9.96 milliseconds each for 39.1 FFT's. This results in a 389.5 millisecond execution and transfer time. Again, allowing for 50 percent overhead safety margin, the total becomes 574.16 milliseconds per second. To perform the discrete filtering would require an additional APGET, APPUT, RFFT, RFFTSC as well as a vector multiply(VMUL) and a complex vector multiply(CVMUL) bringing the time to compute one seconds worth of data to well over one second.

Therefore another AP-120B must be installed to insure that speed requirements are met. Also, since the host computer must be interrupted many times, it may be necessary to retain the PDP-11/34 in the signal processing subsystem. There is also the consideration that if a math routine is custom written, it will not be able to be loaded in the math library which will generate considerable overhead each time it is called. (The amount of this overhead time is system dependent.)

VII. CONCLUSIONS AND RECOMMENDATIONS

The test-bed as proposed seems to be a workable design, although for most applications a more efficient and economical architecture may be constructed.

For many uses the need for the PDP-11/34 computer and the AN-5400 A/D converter seem unnecessary when used in conjunction with the MAP-300 array processor. The Ampex Megastore may be required for a few applications but would not be suitable for the majority of applications (including real-time) since a disk peripheral attached to the PDP-11/70 would be cheaper and still perform the same functions.

It is felt that the increase in complexity and possible confusion using the MAP-300 over the AP-120B can be overshadowed by the reduction in equipment required by the MAP-300. This increased proficiency should even be more greatly felt (assuming a normal learning curve) with subsequent installations. Also, with the time saving in execution, extra calculations could be performed on the MAP in a real-time environment, thereby increasing efficiency, operability and spectrum.

It is recommended that further tests be conducted using the actual applications, data types and speed requirements to fully evaluate the most economical and efficient minimum design necessary.

VIII. REFERENCES

1. A PIPELINED PARALLEL SYNCHRONOUS PROCESSOR FOR HIGH-THROUGHPUT DATA REDUCTION, Floating Point Systems, Inc., FPS-7324, February, 1977.

2. ADAM Model 5120 Analog Data Acquisition Module, CSP Inc., Document number S-13, 1978.

3. AN INTRODUCTION TO THE MAP SERIES MODELS 100, 200, AND 300, CSP Inc., Document number S-02, December, 1975.

4. AP-120B APAL - ARRAY PROCESSOR ASSEMBLY LANGUAGE, Floating Point Systems, Inc., FPS-7275-01, February, 1976.

5. AP-120B APLINK - ARRAY PROCESSOR LINKING LOADER, Floating Point Systems, Inc., FPS-7276-01, February, 1976.

6. AP-120B DEBUG - ARRAY PROCESSOR DEBUGGER, Floating Point Systems, Inc., FPS-7277-01, February, 1976.

7. AP 120B FLOATING POINT ARRAY PROCESSOR, Floating Point Systems, Inc., Form 7244, Revised January. 1977.

8. AP-120B MATH LIBRARY, Part One, Floating Point Systems, Inc., FPS-7288-03, August, 1977.

9. AP-120B PAGE SELECT OPTION (PRELIMINARY), Floating Point Systems, Inc., FPS-7326, February 14, 1977.

10. AP-120B PRICES, Floating Point Systems, Inc., Form 7293 January 76.

11. COMPUTER FAMILY ARCHITECTURE SELECTION COMMITTEE: FINAL REPORT, Vol I, Burr et al, 1 December, 1976.

12. COMPUTER FAMILY ARCHITECTURE SELECTION COMMITTEE: FINAL REPORT, Vol VIII, Clearwaters et al, 1 December, 1976.

13. DATA ACQUISITION UNIT FOR SATCOM SIGNAL ANALYZER, Thesis by Marvin J. Langston, June 1978.

14. HOW TO PROGRAM THE AP-120B, Floating Point Systems, Inc., FPS-7303, March, 1976.

15. INTRODUCTION TO ARRAY PROCESSING WITH SNAP-II, CSP Inc., Document number S-03, Revised March, 1978.

16. IT'S SIMPLE TO PROGRAM SNAP-II ARRAY FUNCTIONS, CSP Inc., Document number S-04, Revised December, 1977.

17. LOOK PROGRAM USER's MANUAL, CSP Inc., Document number DW8004-001-01, June, 1977.

18. MACRO ARITHEMITIC PROCESSOR SYSTEMS (MAP-100/200/300), PROGRAMMER'S REFERENCE MANUAL, CSP Inc., Document number JB6000-001-03, May, 1977.

19. MAP APPLICATION NOTE 1, CSP Inc., Document number MAN-001-01, February, 1978.

20. MAP CROSS ASSEMBLER MODEL 8001 - SOFTWARE INSTALLATION BOOKLET, CSP Inc., Document number DW8001-001-00, May, 1977.

21. MAP I/O Scroll Model 5 Interface Booklet, CSP Inc., Document number DE4080-000-00, May, 1977.

22. MAP Loader in Fortran, CSP Inc., August, 1978.

23. MAP - Macro Arithemitic Processor, Magnavox, FWD77-1591, Fort Wayne, Indiana.

24. MAP MODEL 3 I/O SCROLL (IOS3) INTERFACE MANUAL, CSP Inc., Document number DE6403-000-01, August, 1977.

25. MAP Simulator Program Model 8002 Reference Manual, CSP Inc., Document number JW8002-001-02, October, 1978.

26. MAP SOFTWARE INTERFACE DESCRIPTION FOR THE DEC RSX-11M SYSTEM, CSP Inc., Document number 8901-000-00, July, 1977.

27. MAP STANDARD PRICE LIST, CSP Inc., January, 1977.

28. MINI AND MICRO COMPUTER SURVEY, Datamation, Knottek, August, 1978.

29. PDP-11 END USER PRODUCT SUMMARY, Digital Equipment Corporation, January, 1977.

30. PDP-11/04/34/45/55/60 PROCESSOR HANDBOOK, Digital Equipment Corporation, 1978.

31. PDP-11/70 PROCESSOR HANDBOOK, Digital Equipment Corporation, 1976.

32. PROCESSOR HANDBOOK, Floating Point Systems, Inc., Form 7259-02, May, 1976.

33. PROGRAMMABLE I/O PROCESSOR PIOP, Floating Point Systems, Inc., FPS-7350, June, 1977.

34. SNAP-II Reference Card, CSP Inc., Document number S-11, 1978.

35. SOFTWARE FUNCTIONAL SPECIFICATION - SNAP-II HOST/MAP DRIVER MODULES, CSP Inc., Document number DW6000-006-00, August, 1976.

36. THE AGE OF ARRAY PROCESSING IS HERE, Floating Point Systems, Inc., Form 7345, 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Professor George Rahe, Code 52 Ra Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
5.	Philip Mylet, Code PME-124 NAVELEX Washington, D.C. 20360	1
6.	LT. George T. Vrabel 86 Norseman Drive Portsmouth, Rhode Island 02871	1
7.	Professor John E. Ohlson, Code 62 01 Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1